THE UNIVERSITY *of* EDINBURGH

Computer Aided Design of Printed Wiring Boards

N.A. Rose

Thesis presented for the Degree of Doctor of Philosophy of the University of Edinburgh in the Faculty of Science, April 1970.

Summary

A method is described for the computer-aided layout of printed wiring boards. The type of board considered is a single sided board containing discrete components. The required input for the layout algorithm is coded from the relevant circuit diagram, together with a description of the component dimensions. This information is then stored within the computer in a data structure. The circuit components and their interconnections are represented by a set of nodes and branches.

The principles of graph theory are used to construct an abstract model of the layout. A number of the nodes and branches of the circuit are first used in the construction of a planar graph. A method is then described for inserting the remaining branches into the graph to form a "pseudo planar graph". This represents a set of components and conductor paths which can be laid out on a single sided board without intersections. The number of conductor crossings is thus minimised before the actual layout commences.

An algorithm is then described for automatically constructing a board layout from the pseudo planar graph. The relative interconnections are already known so the placement of components and routing of conductor paths can proceed simultaneously. The layout is therefore constructed in a series of logical steps working across from one edge of the board to the other. This approach contrasts with the more usual methods of layout in which components are placed first, followed by a search for conductor routes.

The layout algorithm is also provided with facilities for man-machine interaction by means of a graphical display and light pen. Interaction allows the user to alter the positions of

components during the construction of the layout. Thus the skill and experience of the user can be combined with the speed and accuracy of the automatic algorithm. Interaction also enables special conditions to be incorporated into the layout which would otherwise entail considerable programming effort.

Three different circuits are used to test the layout algorithm. The results are shown for layouts constructed both automatically and by the use of interaction. One layout is also compared with a manually-produced layout of the same circuit. The results show that a feasible method has been developed for the layout of printed wiring boards by computer. Comparable results are produced in considerably less time than normal layout methods.

Table of Contents

Chapter 1 Introduction

Printed wiring boards are used by nearly all manufacturers of electronic equipment in the construction of their products. The design of a printed wiring board is a process which takes a considerable amount of the designer's time and is prone to errors. It is therefore desirable to develop a computer program which will quickly and accurately design printed wiring board layouts.

The design of a layout is a complex problem and many of the steps are performed intuitively by a human designer. The writing of a computer program to automatically design a layout is thus a difficult task, and many different methods have already been attempted. A method is described here which uses the principles of graph theory in designing a layout. It also enables the user to interact with the computer to improve the results.

1.1 Printed Wiring Boards

A printed wiring board is a thin board of insulating material upon which an electronic circuit is constructed. Each component of the circuit has a number of terminal wires or pins which pass through holes drilled in the board. The electrical connections required to form the circuit are printed onto the surface of the board as a set of copper paths or conductors. The board provides insulation between adjacent conductors but does not allow conductor paths to intersect, except where a connection is intended. The arrangement of component positions and conductor paths on a board is termed a layout.

Printed wiring board layouts may take a number of different forms. Some circuits consist of components of varying types and sizes such as resistors, capacitors, transistors, etc. termed

discrete components. Other circuits consist of integrated circuit components, all the same size and shape and arranged in a fixed matrix of positions on the board. Other circuits still contain a mixture of discrete components and integrated circuits, such as an integrated circuit amplifier with a number of feedback components.

The conductors of a printed wiring board may be placed on one side of the board, termed a single sided board, or on both sides, termed a double sided board. In some cases a board may be constructed as a set of laminations containing as many as twelve layers of conductors. The reason for using more than one layer of conductors is that on a closely packed board there may be insufficient area for all the required conductors on one side of the board. Also, it is often theoretically impossible to route all the conductors of a circuit without intersections on one side of the board. Electrical connections between the two sides of a double sided board are made by copper lined holes through the board, called through-plated holes.

External connections to the printed circuit board may be made by connecting wires to terminal pins on the board. More generally, however, all the external connections of the circuit are brought to a set of gold-plated conductors on one edge of the board, termed the edge connector pins. This edge of the board then plugs into a multi-way socket to make connections to external signals and power supplies.

1.2 Objective of Board Layout Design

The objective of this project is the development of a computer program to lay out printed wiring boards. The type of boards to be considered are single sided boards with an edge connector along one side of the board. The components placed on the board are to be

discrete components of any number of pins, including integrated circuits. There is considerable interest in laying out this type of board as the majority of electronic circuits use discrete components and many circuits may readily be constructed on single sided boards. Single sided boards have an advantage over double sided in that they cost less to produce. The program is not intended to lay out boards containing only integrated circuits in a fixed matrix of positions. This problem is preferably solved by other methods, some of which are described in chapter 2.

Ideally the program should be completely automatic in its operation so as to produce results in a minimum of time. It has, however, been found virtually impossible to specify an algorithm which will satisfy all of the constraints and conditions required by a general purpose layout program. The program therefore includes facilities for human interaction with the layout process by means of a graphical display and light pen. This enables the experience and skill of the user to be included in the design process whilst relieving him of the task of having to accurately draw the detail of the layout.

1.3 General Method of Layout

Most computer methods of board layout already in use divide the problem into two independent parts. These are the placement of components on the board, followed by the routing of conductors. Some of these methods are reviewed in chapter 2. The disadvantage of this approach is that there is no form of feedback from the conductor routing to the component placement stages. There is, for example, no re-arrangement of adjacent components to allow an extra conductor to pass between,such as a

designer would try in a practical case. This often means that the routing algorithm spends considerable time searching for conductor paths which are topologically impossible to complete.

The method of layout described here attempts to resolve this difficulty by constructing a topological representation of the circuit first, before the layout is constructed. This means that during component placement, due space may be allowed for the conductor paths on the board.

As the conductors are to be routed on a single side of the board, all conductor crossings must be eliminated from the layout. A topological model of the layout is constructed by the use of graph theory so as to remove all crossovers. Its method of construction is described in chapters 3 to 6. The circuit topology is then known so the physical layout may be constructed by a series of logical operations. The layout algorithms constructs small sections of the layout in turn, working from one edge of the board across to the opposite edge. Any special constraints required are incorporated into the layout by means of graphical interaction. The construction of the layout and the use of interaction are described in chapters 7 to 9. The results and possible improvements to the method are discussed in chapters 10 and 11 respectively.

1.4 Computer System

The computer system on which the layout program has been developed is described here as it has, in several ways, influenced the manner in which the program has been written. A diagram of the system is shown in Fig.1.1. The computer used for the major part of the computation is an ICL 4130. It has the usual peripherals of paper tape readers and punches, control teletype, line printer and

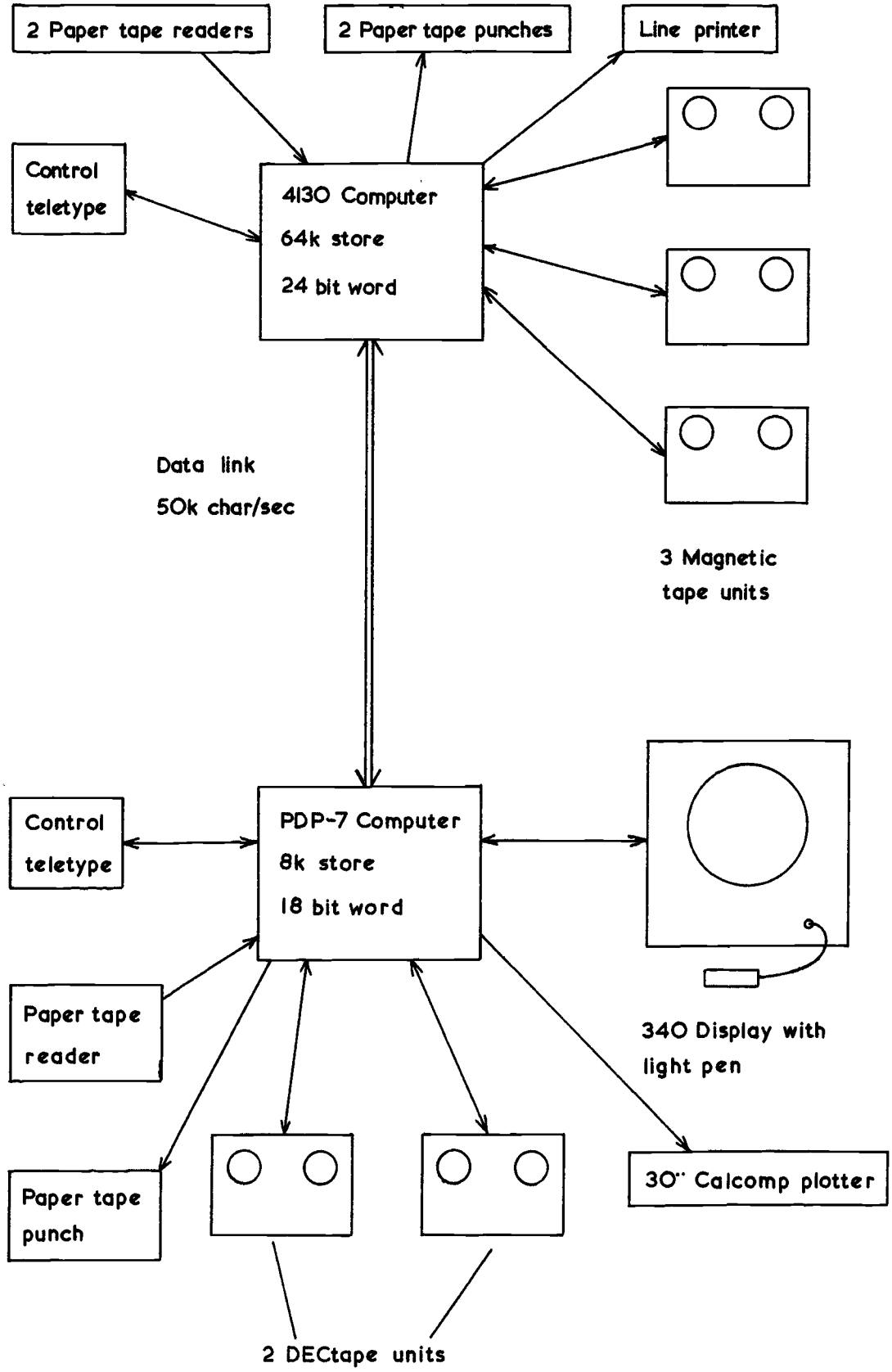


Fig.1.1 Computer Installation

three magnetic tape units. It is connected by a high speed data link to the PDP-7 computer. The link allows communication in both directions between the two computers at rates of up to 50,000 characters/second.

The PDP-7 computer deals with the maintenance of the display and the servicing of light pen interrupts during interaction. The Type 340 display with light pen operates asynchronously with the computer but shares its core store for storage of display file. The useful area of the display is $9\frac{1}{2}$ inches square with a resolution of 1024 points along each axis. The other peripherals of the computer include a paper tape reader and punch, a control teletype, two DECtape magnetic tape units and a 30 inch wide drum type Calcomp plotter.

The ICL 4130 computer is programmed in FORTRAN IV using the magnetic tape based FORTRAN system. Due to the limitations of core space and the maximum allowable number of subroutines, the overall layout program is split into two parts and run as two consecutive programs. These are the topological part and the board layout part. No special programming is required for the PDP-7 computer as previously developed display and interaction software is used (see Appendix B).

A note is appropriate here on the use of the words "topological" and "graphical". "Topological" is used to describe the abstract structure, or graph, of the circuit to be laid out before it has been given physical dimensions. "Graphical" is used to describe the visual display of the board layout whilst it is being constructed. These two words will be used in the sense just given to avoid any confusion of terms.

Chapter 2 Survey of Existing Methods of Layout

This chapter presents a survey of some of the known methods of laying out printed wiring boards by computer. Most methods tend to divide the problem into two separate parts; placement of components followed by routing of conductors. Some topological methods of constructing layouts have been proposed but none of them appear to have been put into practice.

2.1 Methods of Component Placement

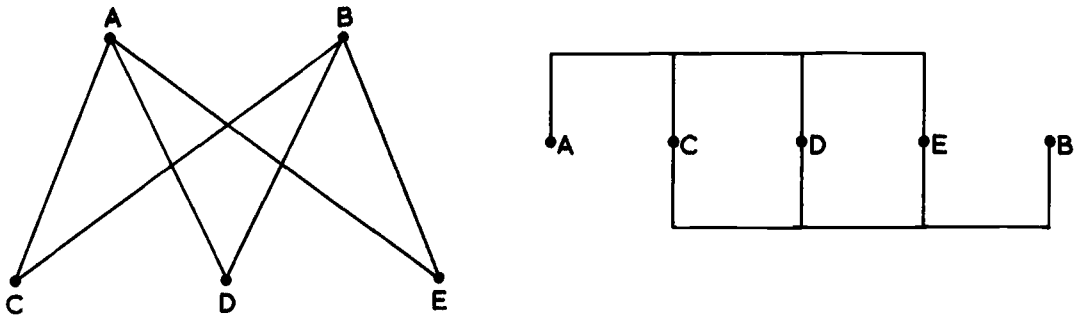
Most of the papers published on board layout tend to concentrate on circuits in which all the components are integrated circuits of the same size and shape. This means that the board may be divided into a fixed matrix of positions such that each component occupies one position. The component placement problem then resolves itself into one of deciding into which position to place each component. The criterion of a good placement is usually taken to be one which gives a minimum total conductor length. The desired result is to reduce conductor congestion on the board and to reduce the effects of capacitance between adjacent conductors. Conductor lengths are assumed to be the point - to - point distances between connected components as the actual conductor paths are not known at this stage.

The method described by Rutman (30) uses the idea of "unconnected sets" of components. The components are given an initial placement on the board, either randomly or manually. They are sorted into a number of unconnected sets such that none of the components within a set are interconnected. An unconnected set of components is then removed from the board and each component in the set is systematically placed in every vacant space on the board in turn. The total length of interconnections between the component

and those already on the board is calculated at each point. As the component belongs to an unconnected set, its length of interconnections is independent of the remaining components in the set. A matrix of all the component positions and conductor lengths for the set is constructed. From the matrix, the optimum placement solution for the set is calculated such that the total wire length is a minimum. The procedure is repeated for all of the unconnected sets. The layout solution is further improved by interchanging the positions of connected components in an attempt to reduce the lengths of the longest wires. The method should result in a compact layout but it is only feasible for circuits in which all the components are of the same size. Also, the method takes no account of the topology of conductor connections.

The method described by Mamelak (22) is used for the placement of computer logic modules. From the logic diagram of the circuit to be laid out, a connection matrix of components, or logic modules, is constructed. The components may be divided into a set of "chains" such that each chain consists of an interconnected group of components, two of which are connected at least to the remaining components of the group. A chain is illustrated in Fig. 2.1(a), where components A and B are the two "vertices" of the chain and the remaining components C, D and E are the "base points" of the chain. One property of a chain is that it may be rearranged as shown in Fig. 2.1(b) to reduce the number of conductor intersections.

A permutation procedure is used to divide the components into a set of chains such that each chain may be placed on one row of positions of the board. The row chosen for each chain depends



(a) Basic chain (b) Placement in row of board

Fig.2.1 Chain of Components

on the total length of interconnections with other chains. Thus the y co-ordinate of each component is determined and the connections between the components of a chain may be made in the x direction on one layer of the board with a minimum number of intersections. A similar procedure is used to assign an x co-ordinate to each component and to reduce the number of intersections between conductors in the y direction on the second layer of the board. Although this method is only suitable for integrated circuit layouts, it does take some account of the wiring topology by attempting to reduce the number of intersections.

The method described by Case (6) is used to assign the positions of small circuit cards on a large "mother board". The method allows an engineer to specify the positions of selected cards. Of the remaining cards, one is selected and tried in every vacant position on the board. The total length of conductors between the card and the already assigned cards is calculated at each position. The card is placed in the position that gives minimum conductor length, and the procedure is then repeated with

each of the remaining cards. It does not give an optimum placement of components so a further procedure is used to improve the placement. The second procedure attempts to interchange the position of each card with every other card on the board. If an interchange results in a reduction of total wire length, the card positions are re-assigned. Again, this method produces a compact layout with low total wire length but does not consider the wiring topology.

The method described by Dunne (7) constructs a layout in stages rather than attempting component interchanges as described in the previous methods. The algorithm selects a location on the board which is nearest to the components which have already been placed. (Initially only the edge connector is placed.) From the list of components to be placed, the one which has the greatest number of connections to the already-placed layout is selected. The algorithm then attempts to route the conductors to the component, using a double sided board. If the routing is not successful, further components are tried in the given board location. If a solution still cannot be found, a new board location is chosen. The procedure is repeated for each component in turn. The method has only been used for integrated circuit components but could possibly be modified for discrete components. It gives the first solution encountered and does not attempt to find the optimum solution. The method does, however, check that all conductor paths can be routed to a component before placing it.

The one method encountered which deals specifically with discrete components is the ACCEL program (9). Component positions are assigned by a "force placement" method. Each conductor of a circuit is assumed to be an "elastic wire" such that it exerts a force of

attraction, proportional to its length, on the component at either end. The effect of this force is to group together components which are closely connected. In addition, forces of repulsion exist between adjacent components to prevent component overlap. The conductors also exert a "torque" upon components in order to select the best orientation for each component.

The components are given an initial arbitrary placement. The program then operates in an iterative manner, summing the forces on each component in turn and moving it towards an equilibrium position. The forces of attraction are initially high and the forces of repulsion low so as to rapidly improve the layout. After a number of iterations, the components are constrained to a vertical or horizontal position, whichever is nearest to the current component orientation. The forces of repulsion are increased to prevent component overlap and the iterations are continued until component movements are negligible. The method thus gives a good placement of different-sized components. Although closely connected components are grouped together to reduce the total conductor length, no account is taken of conductor topology. The method of conductor routing is described in the next section. A similar method is used by Leevers (20) for the placement of integrated circuits. In the final stages of this method, components are forced onto the nearest allowable board positions.

2.2 Methods of Conductor Routing

Nearly all methods of conductor routing start with the assumption that the components have already been placed. The problem is thus one of connecting together pairs of terminals.

The connecting paths must be routed so that no paths intersect and it is desirable that the total conductor length is a minimum. Most methods assume that a double sided printed wiring board is used.

The classic method for constructing conductor paths is Lee's algorithm (19). The board is divided into a grid of squares. Those which contain obstacles such as component terminals or conductors are marked as being occupied. The two squares to be connected together, the start and target squares, are specially marked. All unoccupied squares around the start square are marked with a '1'. All the unoccupied around these are marked with a '2' and so on. A wave of marked squares thus spreads out from the start square until the target square is reached. It is then a simple matter to trace a path back to the start square. The algorithm is generally modified because the search wave spreads in all directions from the start square, involving unnecessary computing time. Secondly, the algorithm will find all the paths of equal length between two points but has no way of distinguishing between the different paths.

The ACCEL method of conductor routing (9) uses a novel topographical model of the layout for routing. The board is divided into a grid of squares, each of which may be assigned an "altitude". Initially all the squares are set to zero altitude. Any obstacles such as component pins or holes in the board are assigned an altitude so as to form a "peak". The edges of the board are represented by a ridge around the layout. To find a path between two pins, the target peak is depressed to a negative altitude. A modified version of Lee's algorithm is then used to find the most downhill path from the start to the target pin.

The program has several phases of operation. Firstly, all paths are routed simultaneously for a number of iterations. Paths successfully completed are inserted as ridges in the topographical model so as to repel conductors routed in later phases and avoid congestion on the board. Secondly,the procedure is repeated with all the remaining conductors routed simultaneously. Thirdly, the procedure is repeated with the remaining conductors, routing one conductor at a time. The method can be used for either single or double sided boards. In the case of a double sided board, the whole procedure is performed on one side of the board, then repeated for the remaining conductors on the second side of the board.

Other modifications may be made to Lee's algorithm in order to improve its efficiency, as illustrated by Mikami and Tabuchi (24). In this method a double sided board is used with all horizontal conductors routed on one side and all vertical conductors routed on the other side. This avoids the problem of crossing conductors but restricts the conductor paths which may be formed. The board is again divided into a grid but instead of searching square-by-square, the search is performed line-by-line. From the start square, four lines, limited in length by existing obstacles, give the possible directions of the search. Each of these lines may pass through the board at a number of through-plated holes. Each of the through-plated holes may therefore be developed into two more lines on the opposite side of the board. The procedure is continued until the target square is reached. The line-by-line method of searching is considerably faster and uses less storage space than Lee's algorithm.

Two methods of conductor routing on double sided boards are described by Kodres and Lippmann (13). The first approach sorts sets

of interconnected pins, or nets, into a list of decreasing net size. The size of a net is defined by the perimeter of the rectangle surrounding all pins in the net. To route a given net, the pins furthest apart are connected by a path which uses the least number of through holes. The remaining pins of the net are then connected onto the path already routed. Paths are only chosen which lie within the rectangle of the net and which use less than a specified number of through holes. These constraints help to reduce board congestion. When all of the nets have been processed, any remaining conductors are routed by using Lee's algorithm to search exhaustively for a path.

The second approach divides the board into a grid of squares and assigns a congestion cost to each square. A square is given a high cost if it can be used by many nets, so that conductor paths will tend to avoid congested parts of the board. The nets are connected one at a time in order of increasing conductor length. For each connection, the path is chosen which has the lowest congestion cost and which uses the least number of through holes. The two methods described avoid the problem of conductor crossings by routing conductors horizontally on one side of the board and vertically on the other side.

A method of conductor routing for double sided boards, using graphical interaction, is described by Leevers (20). A graphical display and light pen are used to display and modify one side of the board at a time. Each conductor is initially displayed as a straight line joining two end points. Near-vertical conductors are assigned to one side of the board and near-horizontal to the other

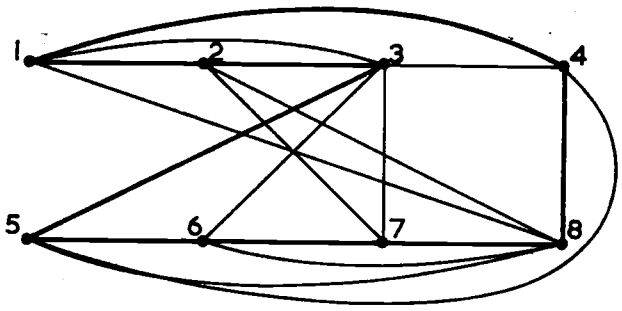
to the other although the assignment may later be altered if desired. Low level routing facilities allow conductors to be routed around obstacles by the insertion of intermediate corners, and diagonal conductors to be replaced automatically by conductors of horizontal and vertical sections. Higher lever facilities attempt to automatically route each conductor in turn by application of the low level facilities. The program initially attempts automatic routing and usually succeeds with many of the conductors. In cases where a path cannot be found, the operator intervenes and uses the low level facilities, by means of the light pen, to re-order part of the layout. The skill of the operator is thus used to assist the program in difficult parts of the layout. In later stages of the layout, the method relies heavily on the operator to find conductor paths.

There are a number of advantages and disadvantages of splitting the layout problem into the separate parts of placement and routing. These are discussed further in Chapter 7.1.

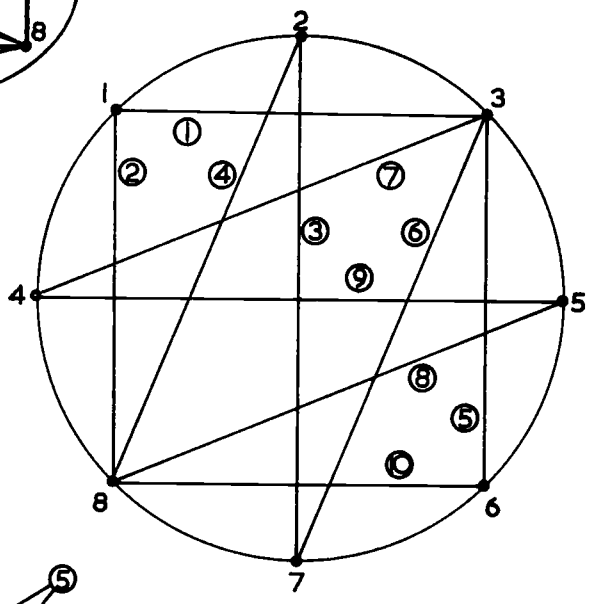
2.3 Topological Methods of Layout

The principle of the topological methods of layout is to minimise either the number of conductor crossings or the number of conductors removed from the layout to eliminate crossings. Several algorithms have been programmed but none appear to have been taken to the stage of actually producing a layout.

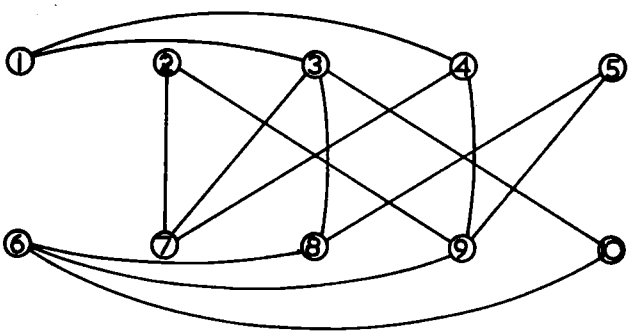
In the method described by Bader (1), the branches of a graph are re-arranged, and some removed, so as to eliminate all crossings. An example from the paper is shown in Fig. 2.2. The graph is searched for a closed circuit which includes as many of the nodes as possible; in this case all the nodes, as shown in



(a) Graph with crossings



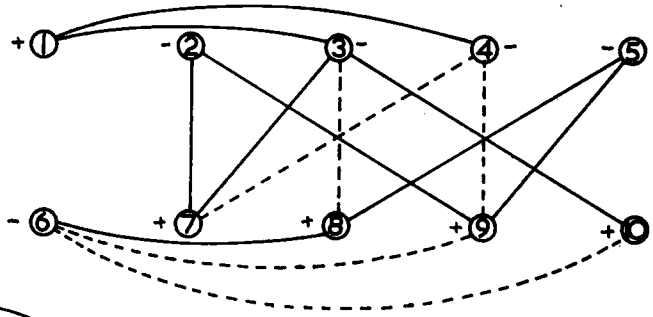
(b) Graph redrawn



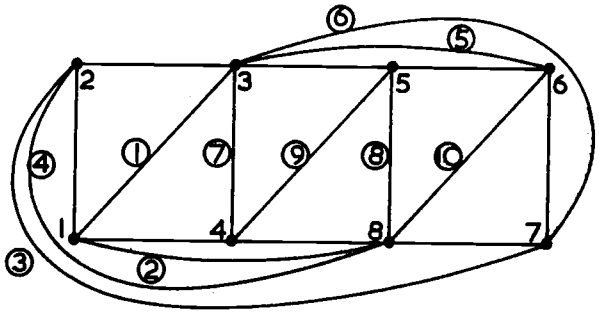
(c) Auxiliary graph

+ inside

- outside



(d) Assignment of branches



(e) Graph with crossings removed

Fig.2.2 Construction of a Planar Graph

Fig. 2.2(a). The graph is redrawn with the circuit on the outside edge and the remaining branches within as shown in Fig. 2.2(b). An auxiliary graph is then drawn, Fig. 2.2(c), whose nodes correspond to the branches on the inside of Fig. 2.2(b). Pairs of branches which conflict in Fig. 2.2(b) are represented by branches joining the corresponding nodes in the auxiliary graph. The branches of Fig. 2.2(b) may be assigned to either the inside or the outside of the closed circuit in order to remove crossings. The assignment is made by starting with an arbitrary node in Fig. 2.2(c), node 1, and assigning it to the inside. Adjacent nodes are then assigned to the outside and so on, as shown in Fig. 2.2(d). If the graph is non-planar, branches are removed at this stage. The graph may then be redrawn without crossings as shown in Fig. 2.2(e). The method has been further developed and programmed for computer by Fisher and Wing (8). A matrix method is used to process the graph so that non-planar branches are identified and removed from the graph.

The algorithm described by Nicholson (26) minimises the number of crossings in a graph, rather than deleting non-planar branches. In this method, the nodes of the graph represent components and the branches represent interconnections. The nodes are arranged in a straight line and the branches are drawn as semicircles above or below the node line as shown in Fig. 2.3. The graph may then be described by a permutation of the order of nodes and the direction of the branch semicircles. An initial permutation is constructed by selecting an initial node then adding the node which has the most connections to the existing part of the permutation. This is repeated for all nodes, inserting each one into a position which gives least crossings. An iterative procedure

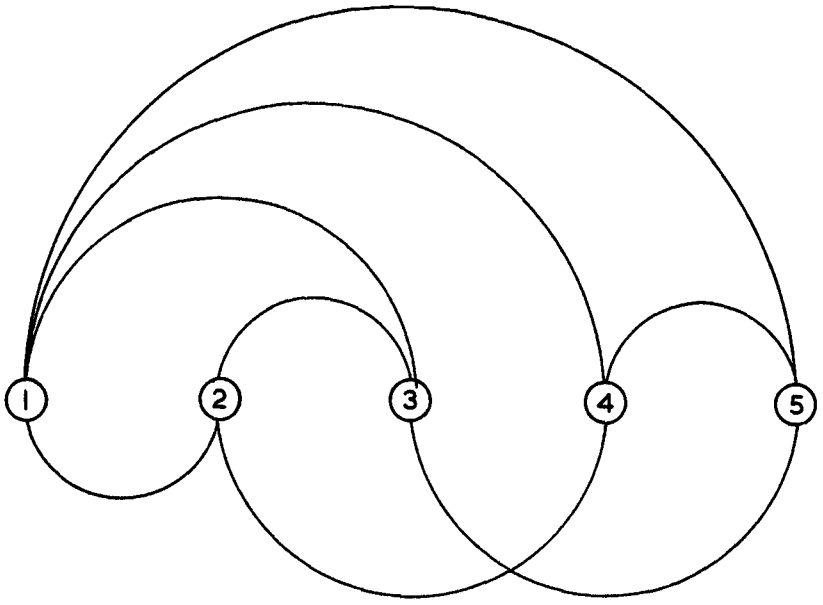


Fig.2.3 Permutation Representation of a Graph

then changes the order of nodes in the permutation in an attempt to further reduce the number of crossings.

A method of constructing a planar graph of components and interconnections is described by Rowley (29). The circuit is defined by a list of components and a list of interconnections. A set of branches is selected so that a "tree" of all the components may be constructed. Each new component added to the tree is connected by one branch only as shown in Fig. 2.4. A "tree list" of all the component pins in order around the tree is made as shown by the dotted path in Fig. 2.4. Any other interconnection in the circuit will divide the tree list into two parts at the points of connection. Two branches are in conflict if the two parts of the tree list formed by one branch each contain a node of the other branch. A matrix of all the branch conflicts is then constructed. From the matrix, a set of conductors is selected such that the number of non-planar branches removed from the graph is a minimum. The resultant graph is not necessarily the optimum planar graph as it is dependent on the branches selected for the

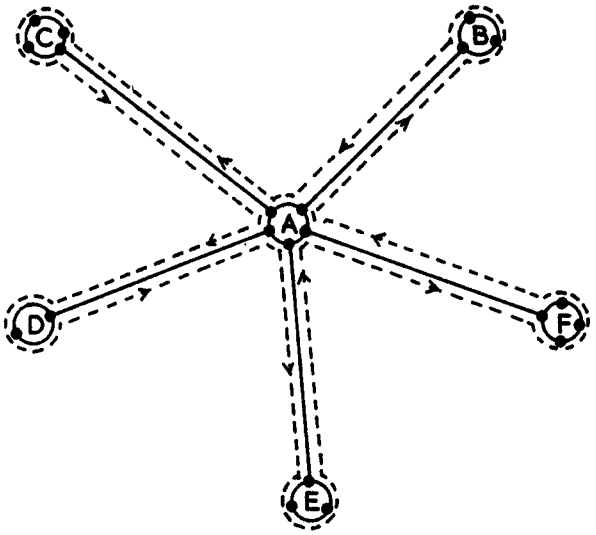


Fig.2.4 Formation of Tree List

initial tree.

The advantages and disadvantages of the topological methods described above are discussed further in Chapter 4.2.

Chapter 3 Topological Representation of a Circuit

Any computer program that generates a printed wiring board layout must have a method of representing the layout within the computer. This chapter describes how the topological representation of a circuit is constructed from the computer input data. The representation is later used to construct a topological model of the layout. The ways in which the circuit representation is actually programmed are described in Chapter 6.

3.1 Requirements of the Topological Representation

The groups of data required for the construction of a layout are the circuit diagram, the physical dimensions of the components and the dimensions of the board. The circuit diagram basically describes the types of components used and the way in which they are interconnected. The information on the circuit diagram should therefore be coded into a suitable format for input to the computer.

The first part of the layout method described here deals with the construction of a topological model of the layout. In developing this model, the circuit is investigated for planarity by examining the way in which components are connected together. The topological representation should therefore indicate the order in which components are connected, without being concerned with the physical co-ordinates of components and conductors.

There are two widely-used methods of representing, within a computer, the interconnections of a graph. The first method is a matrix representation of the graph. Usually, the rows and columns of the matrix represent the nodes and branches respectively. Each element of the matrix is then marked to indicate the incidence, or non-incidence,

of a given node and branch. The second method of representing a graph uses a ring data structure in which data blocks are used to represent nodes and branches. Pointers between the data blocks indicate the interconnections between nodes and branches.

The method of representation chosen for the layout algorithm is the ring data structure. Although an electronic circuit often has a large number of nodes and branches, there are generally few branches connected to each node. A matrix representation would therefore require a large matrix in which most of the elements were empty. A data structure provides direct pointers from, say, a branch to its two nodes. To obtain the same information from a matrix, the whole branch column of the matrix would have to be searched. A further advantage of the data structure is that additional data such as component name, type of branch, or display file may readily be associated with each data block.

3.2 Elements of the Topological Representation

The graph of an electronic circuit is constructed from a number of different types of nodes and branches. Circuit nodes have a corresponding node in the graph but components have a different representation depending on whether they have two, or more, pins. The circuit elements and their corresponding graph representations are described below.

3.2.1 Circuit Node

A circuit node is a point of common electrical connection of two or more components. The corresponding node in the graph has no physical representation. It merely fulfils the function of listing all the components connected to a common point, or to a given

conductor on the board.

3.2.2 Branch Component

A component with two connecting wires or pins such as a resistor or a capacitor is termed a branch component. It is represented in the graph by a component branch and is connected between two nodes. Each of the nodes is the abstract representation of a connection. A component branch is therefore physically equivalent to the component together with part of the conductor paths at each end of the component.

3.2.3 Subgraph Component

In representing a component with more than two pins, such as a transistor or an integrated circuit, several problems arise. The first problem is due to the physical dimensions of the component and the fact that each component pin is connected to a circuit node. In constructing a planar graph of a circuit it may happen that a number of branches, or conductors, have to pass between two particular nodes. If the nodes are connected to two pins of a component, it is quite probable that there would be insufficient space for the conductors to physically pass between the pins. To prevent such an occurrence, each pin of the component is connected to its two adjacent pins by a pseudo branch. Also, each pin of the component is represented by a subgraph node so that every pseudo branch is connected between two subgraph nodes. The pseudo branches initially prevent any conductors from passing between the component pins and they keep all the pins of the component together in a closed planar region. Because of these functions, pseudo branches may never be removed from the graph.

Assuming that multi-pin components are represented by a ring of pseudo branches, a planar graph of the circuit could be constructed, containing these components as subgraphs. A second problem which is not resolved by some other methods is that all of the subgraph components must have the same orientation in the planar graph. The physical analogy is that all the components are mounted on the same side of the board. Defining a component as a ring of pseudo branches readily enables the orientation of a component to be checked during the planarity algorithm.

The third problem in representing a multi-pin component lies in the deletion of non-planar branches between closely connected components. An example of two closely connected integrated circuits is shown in Fig. 3.1. A connection has to be removed to make the

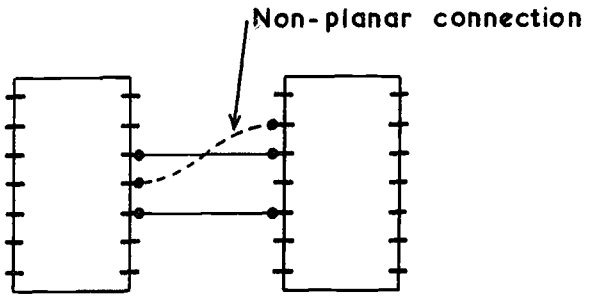


Fig. 3.1 Closely connected components

graph planar. No pseudo branches can be removed however, because the planarity of the component pins would be lost. Each subgraph node has a link branch connecting it to the corresponding circuit node. The physical representation of a link branch is a length of conductor connecting the component pin to the rest of the circuit node. In the event of a non-planarity, one or more of the component link branches may be removed. The circuit diagram of a transistor

and its corresponding topological representation are shown in Fig. 3.2 as an example.

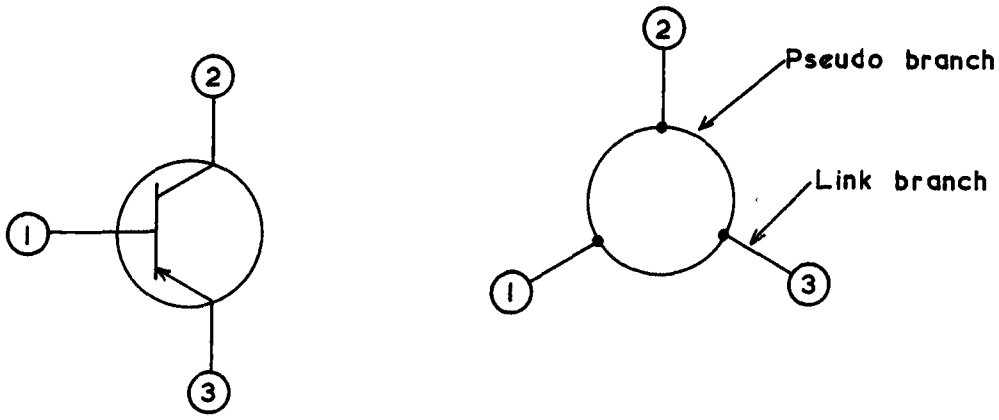


Fig. 3.2 Representation of a subgraph component

The complete set of subgraph nodes, pseudo branches and link branches for a component is termed a subgraph component, or subgraph. The nodes and branches of a subgraph are represented by a set of node and branch data blocks. They could be represented more compactly by a single data block. The graph structure would then no longer be compatible with circuit nodes and branch components however, and programming would thus be more difficult.

3.2.4 Edge connector

The edge connector of a printed wiring board consists of a row of pins or terminals along one edge of the board. As no conductors can pass between the edge pins, their corresponding nodes in the graph must be adjacent to each other. This is ensured by connecting the nodes into a path by a series of pseudo branches, the order of nodes in the path being the same as the order of edge pins on the board. A further pseudo branch is connected between the first and last nodes of the path, thus forming it into a closed loop. This pseudo branch therefore represents the outside edge of

of the board, apart from the edge connector. The closed loop of pseudo branches serves as a boundary within which the topological model must lie.

3.3 Circuit Data Input

The first step in generating a layout is to prepare the circuit data in a suitable format for input to the computer. Two groups of data are required for the construction of the topological model of a layout. The first is a library of component data which may be common to all circuits laid out. The second is a list of components and their interconnections for the particular circuit to be laid out.

3.3.1 Component Library

When constructing a layout, certain data is required for each component, such as its physical dimensions and its number of connecting pins. Most circuits contain several instances of each different type of component. The most economic way to describe the components therefore is to give each one a type number then associate one full description of a component with each different type number. In other words, a library of component descriptions is generated. In a manufacturing organisation this information would probably be stored as part of a data bank which would hold a list of all the types of components ever used together with their electrical and physical characteristics. For the purposes of the method described here, an elementary component library is associated with the circuit data.

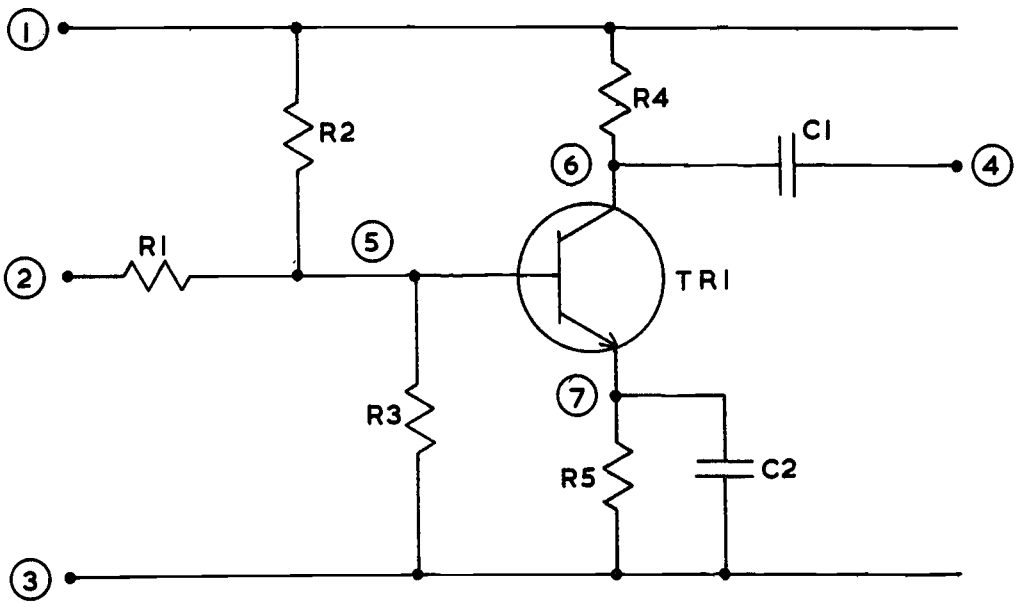
The data for each type of component is stored in a master component block. The master component blocks are held together in a list and each one is given a unique name. For example, the blocks representing $\frac{1}{4}$ watt resistors, $\frac{1}{2}$ watt resistors and transistors may

be called RES1, RES2 and TRAN respectively. The size of a component is defined by a boundary rectangle. It allows space for the component itself, space for connections and fixings and clearances for component spacing. This simplifies board layout as component rectangles may then be placed directly adjacent to each other without further computation of clearances. A master component block stores data on the dimensions of the component rectangle, the number of component pins and the co-ordinates of each pin relative to the component rectangle. Several dummy master component blocks are used to indicate certain functions during data input. These are described in the next section.

3.3.2 Preparation of Circuit Data

To prepare data from a circuit diagram, each electrical node is first labelled with a unique positive integer. A simple example is shown in Fig. 3.3(a). The connections of each component may therefore be described by listing the nodes to which it is connected. The correct orientation of component connections is ensured by adopting a convention of node numbering. Two pin components with a marked pin of polarity such as diodes or electrolytic capacitors are listed with the marked pin as the first node number. Multi-pin components have their pins ordered in a clockwise direction, looking from the conductor side of the board. The first pin in the node list corresponds to the first pin co-ordinate in the master component block.

To code the data from a circuit diagram, each component is described by its name and a list of its node numbers. An example of data coding is illustrated in Fig. 3.3(b); the component library is not shown. Components of the same type are listed consecutively in



(a) Circuit diagram

```
RES
R1      2       5
R2      1       5
R3      3       5
R4      1       6
R5      3       7
R       1       1
CAP
C1      4       6
C2      3       7
C       1       1
TRAN
TR1     6       5       7
TR      1       1       1
EDGE
        1
        2
        3
        4
        1
STOP
```

(b) Data input format

Fig. 3.3 Preparation of circuit data

a group. Each group is preceded by the name of its master component block and is terminated by a dummy component with negative node numbers. This method is used because of the difficulties of reading in data under FORTRAN FORMAT statements. Two additional dummy master component block names are used. The name EDGE indicates that the following node numbers are the nodes of the edge connector, in the correct order. The list of edge nodes is terminated by a negative node number. The name STOP indicates that all of the circuit data has been specified. An example of a topological representation of the circuit shown in Fig. 3.3 is illustrated in Fig. 3.4.

3.4 Data Input Subroutine

A FORTRAN subroutine called DATAIN has been developed to read in the component library and circuit data and to construct the corresponding data structure. The flow diagram of the subroutine is shown in Fig. 3.5 and the type of data structure constructed is described in detail in Chapter 6.

The subroutine starts by reading in the component library data. As each component type is read in, a new block is created and added to the list of master component blocks. The master component name, the number of pins and the component dimensions are stored in the block. This is followed by a list of the pin co-ordinates. A master component with one pin is used to indicate the edge connector. One with no pins is used to signify the end of the library data and its name, STOP, indicates the end of the circuit data.

Following the component library, the name of the next component group is read in. The component library is searched to find the master component block with the same name. The block then gives the

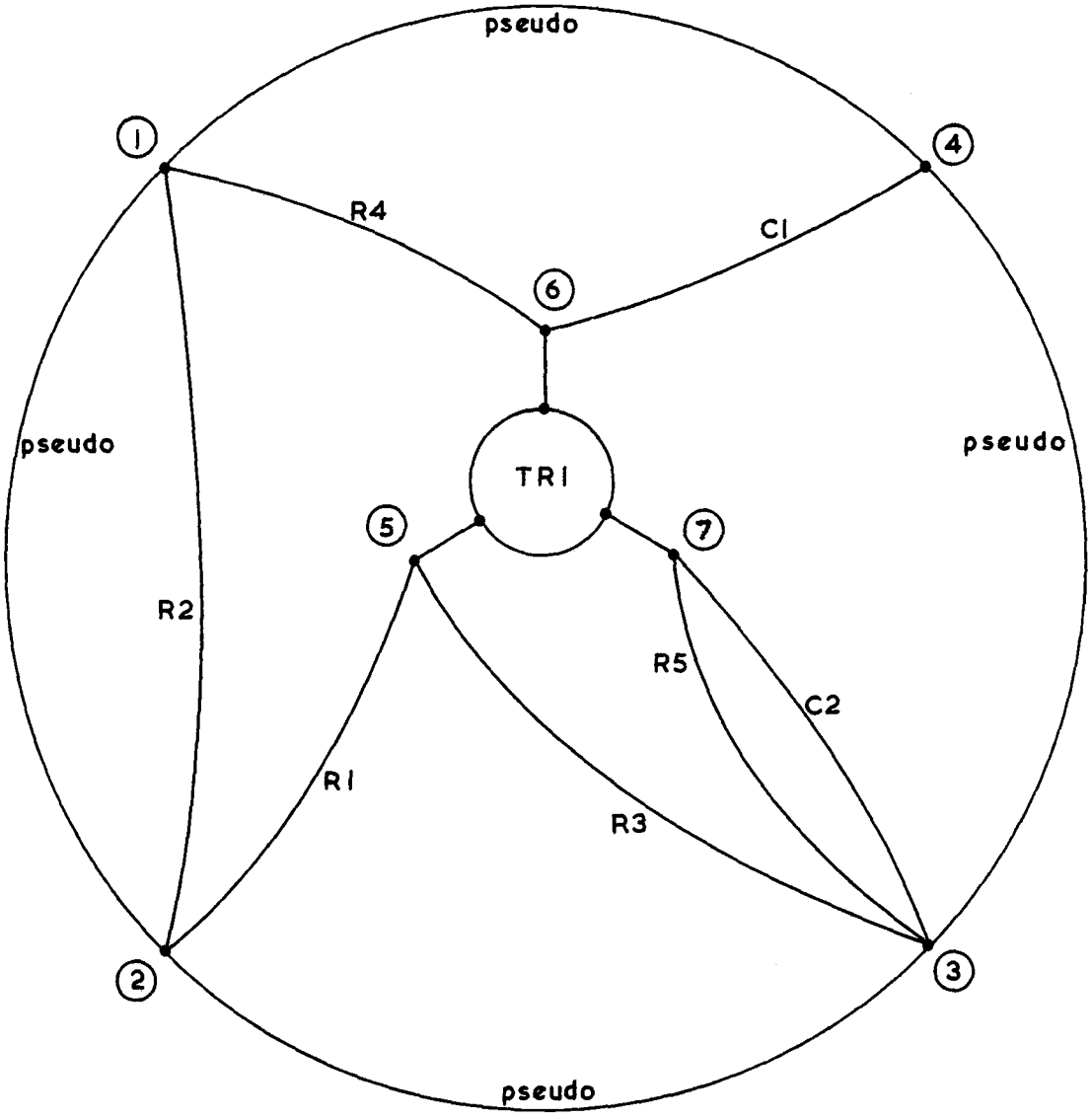


Fig. 3.4 Topological representation of circuit

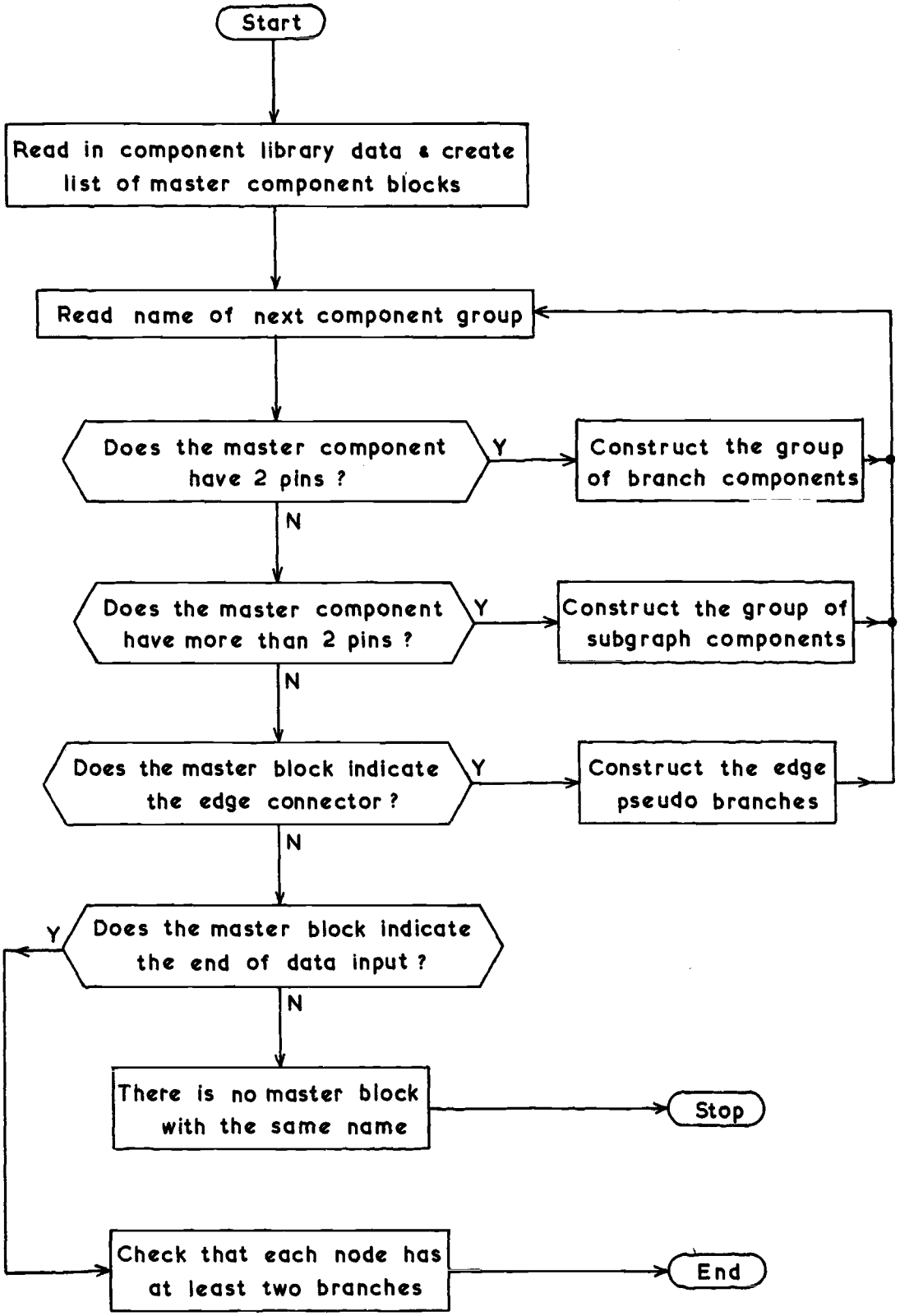


Fig. 3.5 Flow diagram of data input subroutine

number of pins of each component in the group. If no corresponding master component block can be found, an error message is printed out and the program stops.

For groups of two pin components, each component name and its two node numbers are read in turn. A list of nodes is kept in the data structure and this list is searched to find the two nodes of each component. If either of the nodes does not yet exist in the list, a new node block is created and added to the list. The data blocks for a branch component are then created and linked to the existing data structure in the manner described in Chapter 6. The procedure is repeated with each component until the dummy component at the end of the group is encountered. The subroutine then reads in the name of the next component group.

For groups of components with more than two pins, the subroutine reads in the name and appropriate number of nodes for each component. Any new nodes are added to the node list. The required data blocks of subgraph nodes, pseudo branches and link branches are then created for the component and linked into the existing data structure. The subroutine is designed to deal with subgraph components of any number of pins. The appropriate number of pins is merely obtained from the component library. The procedure is repeated for each component in the group until the dummy end component is encountered.

The component group name called EDGE indicates that the next group of numbers is a list of edge connector nodes. The node numbers are read into an array until the dummy end node is reached. Each node is then connected to the next by a pseudo branch in the data structure. The last node is connected to the first by a further pseudo branch. The subroutine then reads in the next component group name.

The component group name called STOP indicates that the circuit data input is complete. An elementary check on the data is then performed. Each node in the circuit node list is checked to ensure that it has at least two connected branches. Any node which has only one connected branch causes an error message to be printed. This check detects some coding and typing errors. The data structure now contains all the data related to the interconnection of circuit components.

Chapter 4 Construction of Planar Graph

The type of board layout considered consists of a set of components placed on one side of the board, a set of conductor paths on the second side and a set of connection pins along one edge of the board. The main objective in producing a board layout is to arrange the components and their interconnecting conductors so that no conductor paths intersect. It has already been shown that a graph may be developed to represent the interconnections of a circuit. This chapter describes an algorithm by which the branches of the graph are ordered, and some removed, so as to produce a planar graph with no branch intersections. Chapter 5 then describes a method by which the non-planar branches are inserted back into the graph.

4.1 Planar Graph Constraints Due to Board Layout

A planar graph is defined as one which may be drawn on a plane in such a way that its branches intersect only at their end points. The plane which is of interest in the board layout problem is the conductor side of a printed wiring board. It therefore follows that the graph representing a circuit must be planar to avoid the intersection of conductors in the physical layout.

When using the graph of a circuit as the topological model of its board layout, a number of problems arise. The first major problem is that the graph of a circuit is seldom planar. A non-planar graph can only be made planar by removing a number of branches although there are usually many alternatives in deciding which branches to remove. A set of branches, preferably a minimum number of branches, has therefore to be identified and removed from

the graph in order to make it planar. The second major problem is that a graph is a topological entity and that planarity is an internal property of the graph. This means that a graph may be given any number of geometrical representations by drawing it on a plane. Having ensured that a graph is planar therefore, the problem still remains in constructing a geometrical representation which has no branch intersections.

The requirements of representing a board layout impose further constraints on the processing of the original graph and on the construction of a planar graph. These constraints are discussed below.

(a) Only component branches and link branches may be removed from the graph in order to make it planar. Pseudo branches must remain in the graph to hold the pins of subgraph components in their correct order and spacing.

(b) All the subgraph components must be connected into the planar graph in the same orientation. This corresponds to all the components being placed on the same side of the printed wiring board.

(c) The nodes and pseudo branches of the edge connector represent the outside edge of the board. They should therefore lie on the outside edge of the planar graph.

(d) The connection pins on each component are connected together in the graph by either one component branch or several pseudo branches. This prevents conductors from passing between adjacent component pins in a planar graph representation of the layout. In the physical layout however, it is possible

for a limited number of conductors to pass between adjacent pins, depending on the component and conductor dimensions. This limitation on conductor paths in the planar graph eliminates the problem of checking clearances between adjacent pins although it usually causes a greater number of non-planar branches to be removed. The constraint is later relaxed and the non-planar branches re-inserted into the graph by the method described in the next chapter.

4.2 Methods of Constructing a Planar Graph

Classical graph theory concentrates on finding the conditions necessary for a graph to be planar rather than devising methods for constructing such a graph. The elegant theorem due to Kuratowski (16) states that a graph is planar if, and only if, it contains neither of the two graphs shown in Fig. 4.1 as subgraphs. The

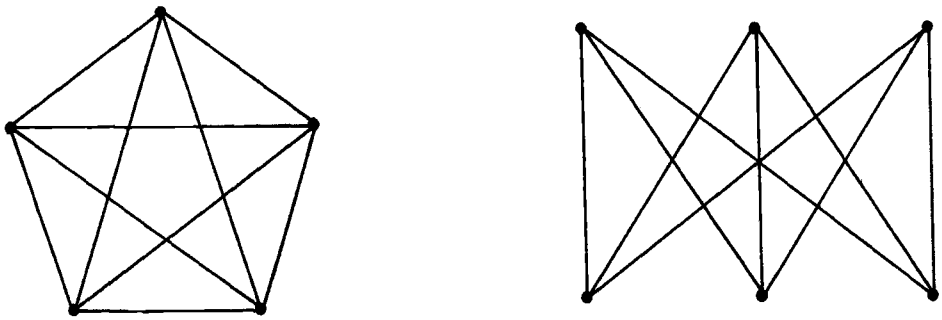


Fig. 4.1 Kuratowski subgraphs

Kuratowski subgraphs may be well hidden within a graph so it is not practicable to search for them in a graph of many nodes and branches. Whitney (31) proves that a necessary and sufficient condition for a graph to be planar is that it has a dual graph. Again, this offers no practical solution to the construction of a planar graph.

A planar graph drawn on a plane without branch intersections divides the plane into a number of non-overlapping regions. Each region is bounded by a circuit, or closed path of branches. MacLane's theorem (21) states that a graph is planar if, and only if, the graph contains a complete set of circuits such that each branch appears in no more than two of the circuits. This theorem is used in the planarity algorithm described in the next section.

Several other methods of constructing the graph of an electronic circuit have been described in Chapter 2.3. These methods suffer several disadvantages, however, in the practical case of producing a board layout. The algorithm for constructing a planar graph described by Bader works satisfactorily for branch components. It is, however, difficult to implement with subgraph components due to the need to preserve correct component orientations. In addition, it is not possible to arrange all the nodes of the edge connector on the outside edge of the graph.

The method due to Nicholson uses a permutation procedure to minimise the number of crossings in a graph. Each component, however, is represented by a node in the graph so that with multi-pin components it is not possible to select the correct order of connections to each component. Rowley's algorithm is particularly suited to circuits containing multi-pin components although part of the procedure involves setting up a matrix for all conflicting branches. This can lead to excessive computer storage and time requirements for a large circuit.

4.3 Principle of Planarity Algorithm

The objective of the planarity algorithm is to construct a planar subset of the graph representing an electronic circuit.

The planar graph should contain no branch intersections and should be subject to the constraints described in section 4.1. Non-planar branches are removed from the graph as they are encountered and no attempt is made to minimise the number of non-planar branches removed. This approach simplifies the planarity algorithm and is justified because non-planar branches are re-inserted into the planar graph at a later stage.

An important assumption upon which the planarity algorithm depends is that every node of the graph is of order two or more. This means that the planar graph may be described by a set of closed paths of branches, each path being the boundary of a planar region. The following circuit and topological conditions show that the assumption is valid for the graph representing a circuit.

(a) Every circuit node, except the edge connector pins, connects at least two components together.

(b) Each edge connector node is connected by two pseudo branches to its adjacent edge nodes.

(c) Similarly, un-used pins on multi-pin components are connected by pseudo branches to their adjacent subgraph nodes.

(d) Separate circuits or components on the board have the edge connector pseudo branches in common with the remainder of the circuit.

4.3.1 Processing of Planar Graphs

Given a planar graph, G, the planarity algorithm is required to re-arrange G into a second planar graph, H. The nodes and branches of graph H are to be ordered so that a geometrical representation

of the graph may be drawn without branch intersections. Graphs G and H have a one-to-one correspondence between their nodes and branches. The difference is that additional information in graph H enables the required geometrical representation to be drawn. Graph H is constructed as a series of subsets of its nodes and branches. An initial set of nodes and branches is chosen so that a planar region is formed, with no branch crossings. Subsequent subsets of the graph are constructed by adding further planar regions to the previous subset such that no branch crossings are introduced.

A path that is known to form a planar region with no branch crossings is the set of pseudo branches representing the edge connector and the outside edge of the board. This path is termed P_1 and is used to form the initial subset of graph H, i.e.,

$$H_1 = P_1$$

When the elements of this subset, or any subsequent subset, are subtracted from graph G, the nodes and branches remaining in G are termed free nodes and free branches respectively. The outside edge of path P_1 forms the boundary of the first planar region of graph H. The region on the inside edge of P_1 is termed the free region as it contains all of the free nodes and branches from graph G which have not yet been defined as part of graph H. In the general case of the nth subset of graph H,

$$\text{Contents of Free Region} = G - H_n$$

Each branch on the edge of the free region has previously been defined as part of a planar region which is adjacent to the outside edge of the free region. Applying MacLane's theorem to the planar graph, there must be a second planar region, on the inside

edge of the free region, which is adjacent to the given branch. The boundary of this second region is defined by a closed path, P_n, which includes the given branch. This path will be comprised of a number of free branches together with part of the free region edge. The node at which the path leaves the edge of the free region is termed the start node; the corresponding node where it returns to the free region edge is termed the target node. It follows that the start and target nodes each lie on the edge of the free region and each have at least one attached free branch.

To add a planar region to the graph H_n, an arbitrary node on the edge of the free region, with a free branch attached, is selected as a start node. The next node on the edge of the free region with a free branch attached is selected as the target node. Starting from the free branches on the start node, a search is made to find the shortest path through the free region to the target node, P_{n+1}. The shortest path is defined as the one with a minimum number of free branches. The path P_{n+1} is then joined to the start and target nodes to form a new planar region. The boundary of the new region consists of one side of the path together with the edge of the free region between the two nodes. The remainder of the free region edge and the second side of the path are redefined as the new free region edge. A new subset of the graph H is thus defined by:

$$H_{n+1} = H_n + P_{n+1}$$

Repeating the procedure with each node on the edge of the free region in turn yields further planar regions of the graph H. The algorithm is terminated when there are no remaining free branches. The free region itself then becomes the final region to be added to the graph.

4.3.2 Example of Planar Graph Construction

A simple example of the planarity algorithm operating upon a planar graph is shown in Fig. 4.2. The initial geometrical representation of the graph shown in Fig. 4.2(a) contains a branch crossing. The objective of the algorithm is to produce a geometrical representation of the graph with no branch crossings as illustrated by Fig. 4.2(b).

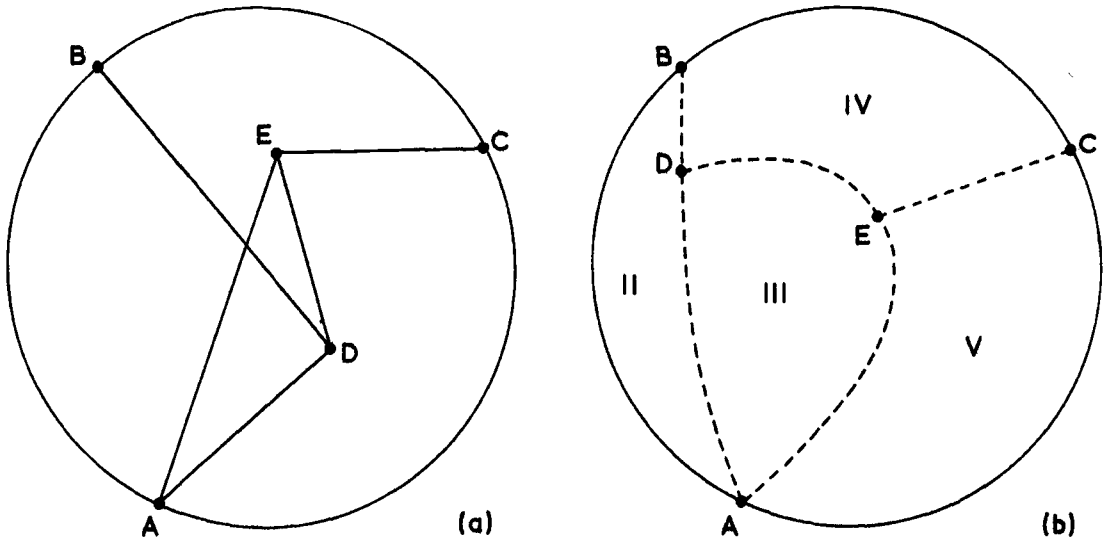


Fig. 4.2 Construction of planar graph

The path of pseudo branches representing the edge connector, ABCA, is taken as the first planar subset of the graph. The outside edge of the path forms the boundary of the first planar region, region I. The inside edge of the path forms the boundary of the initial free region. Node A is arbitrarily selected as the first start node and node B as the target node. A search through the free region for the shortest path from A to B gives the path ADB. Region II is thus defined as the next planar region of the graph and the edge of the free region is redefined as the path ADBCA. Node A remains the start node because it still has a free branch, AE, attached. Node D is then selected as the new target node

nearest to the start node. A search through the free region gives AED as the next path between start and target nodes, giving ADEA as region III.

All of the branches connected to node A are now defined as parts of the planar graph so a new start node, E, is arbitrarily selected from the edge of the free region. The next node on the edge of the free region with any free branches, node C, is chosen as the new target node. The shortest path through the free region from nodes E to C consists of the single branch EC. The edge of the free region between the start and target nodes, DBC, together with branch EC therefore form region IV. The edge of the free region is then defined as CAEC. As there are no free branches remaining in the free region, CAEC becomes the final region V. The branches of the planar graph are thus ordered so as to eliminate all branch crossings.

4.3.3 Processing of Non-Planar Graphs

It will generally be found that the graph of a circuit contains a number of non-planar branches. There are several different strategies for removing such branches from a graph in order to make it planar. One strategy used both by Bader (1) and by Rowley (29) involves making an exhaustive search for all branch conflicts in the graph. From the list of conflicting branches, an optimum set of non-planar branches is selected such that the number of branches removed is a minimum.

A second strategy, which is used here, deals with each branch conflict as it is encountered. When two branches are found to conflict, one of them is immediately removed from the graph although the result will not generally give a minimum set of non-planar

branches. A branch may be unnecessarily removed from the graph if all the branches it conflicts with are themselves later removed. As another algorithm is later used to insert non-planar branches back into the graph, selection of an optimum set of non-planar branches is not critical. The main advantages of this strategy are its speed and simplicity of computation. Each branch conflict is resolved as it is found, instead of having to process a list of many conflicts. Also, in searching the free region of the graph for further planar regions, the number of free branches to examine becomes progressively smaller as more regions are defined.

Any free branch, or path of free branches, that crosses the free region divides the edge of the free region into two parts, E_1 and E_2 at the nodes of connection. A conflict of branches occurs when a second branch or path crossing the free region has one end connected to part E_1 and the other end connected to part E_2. In such a case it follows that the nodes on the edge of the free region, adjacent to the start node of the first path, will belong to the second path. There cannot therefore be a planar path between a start node and either of its two adjacent target nodes.

An example of conflicting branches is shown in Fig. 4.3. Branches AC and BD are in conflict as no path exists within the free region from the starting node A to either of its adjacent target nodes B and D. Neither branch may be drawn around the outside edge of the free region, ABCDA, as the outside has already been defined as part of a planar graph. One of the two branches therefore must be removed in order to make the graph planar.

The algorithm for creating a planar subset of a non-planar graph is an extension of that described in section 4.3.1. A search

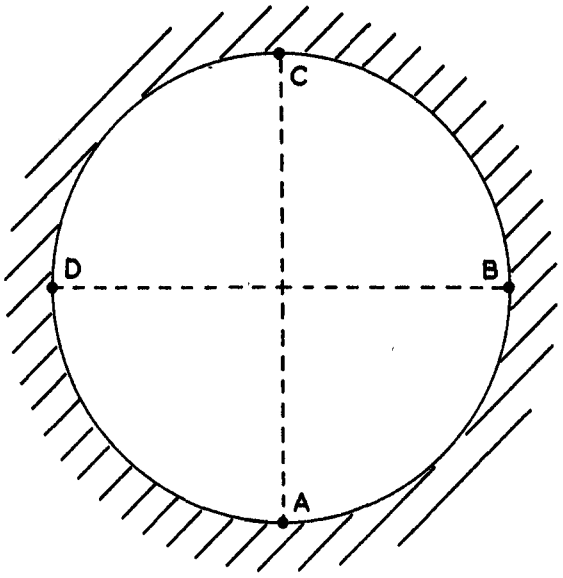


Fig. 4.3 Non-planar branches

is made for planar paths from a start node to each of its two adjacent target nodes in turn. Every time a new planar region is defined, the free region is redefined, a new target is defined, and the search procedure is repeated. Any free branches remaining on the start node that do not yield a planar path after a search are either non-planar branches or bridge branches. Non-planar branches are immediately removed from the graph; the procedure for dealing with bridge branches is described below.

A bridge branch is defined as a branch which is the only connection between the edge of the free region and a subset of the graph G which has not yet been defined in the graph H. This state occurs when successive connections to the subset are removed as non-planar until only the bridge branch connection remains. It is essential that no subset of graph G becomes completely disconnected from the remainder of the graph. If this were to happen, the search procedure for constructing the planar graph would never encounter

the subset by searching from the edge of the free region. The subset would thus not be defined as part of the required planar graph. This same reason also explains the fact, mentioned at the beginning of section 4.3, that every node of the graph must be of order two or more. A bridge branch is thus inserted into the planar graph to prevent a subset from being completely disconnected from the rest of the graph.

An example of the detection of non-planar branches and bridge branches connected to a node is shown in Fig. 4.4. Node A is taken

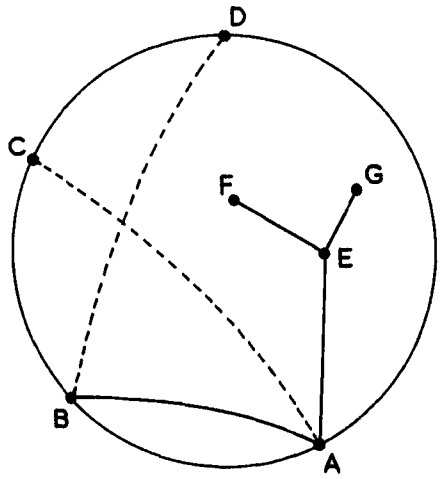


Fig. 4.4 Detection of non-planar and bridge branches

as the start node and its two adjacent target nodes on the edge of the free region are B and D. The first search from node A to target node B yields branch AB as a planar branch so it is inserted into the graph. Two further searches to targets B and D do not yield planar branches so the branches AC and AE on node A must be either non-planar or bridge branches. In order to determine which type they are a search is made from the end of each branch in turn to see if a path exists to any other node on the edge of the free region. If a path does exist, as in the case of branch AC, the branch must be in conflict with another so it is removed as non-planar. If a path does not exist, as in the case of AE, the branch represents the only connection to a particular subset of the graph

so it is retained in the graph as a bridge branch.

4.3.4 Insertion of Subgraph Components

Each planar region of a graph is defined by an ordered ring of branches around its boundary. The method of definition is described in detail in Chapter 6. By convention, the branch order around every region is described in an anticlockwise direction. A subgraph component consists of a planar region bounded by a ring of pseudo branches and by convention these are also defined in an anticlockwise direction.

In searching through the free region of a graph for a planar path, the target node is always arranged by convention to be in an anticlockwise direction around the edge of the free region from the start node. When a subgraph node is encountered, the search proceeds only along the pseudo branch in a clockwise direction from the subgraph node. The conventions of region definition and search direction thus ensure that all subgraph components are inserted into the graph with the same orientation.

4.4 Description of Planarity Subroutine

A subroutine, called PLANAR, has been written to implement the planarity algorithm; its flow diagram is shown in Fig. 4.5. The subroutine starts by connecting the pseudo branches of the edge connector into a closed path. The outside edge of this path bounds the first planar region of the graph and the inside edge of the path is the boundary of the initial free region. The method of linking the branches into a region is detailed in Chapter 6. An arbitrary node with free branches attached, on the edge of the free region, is selected as the first start node. The next node with free branches

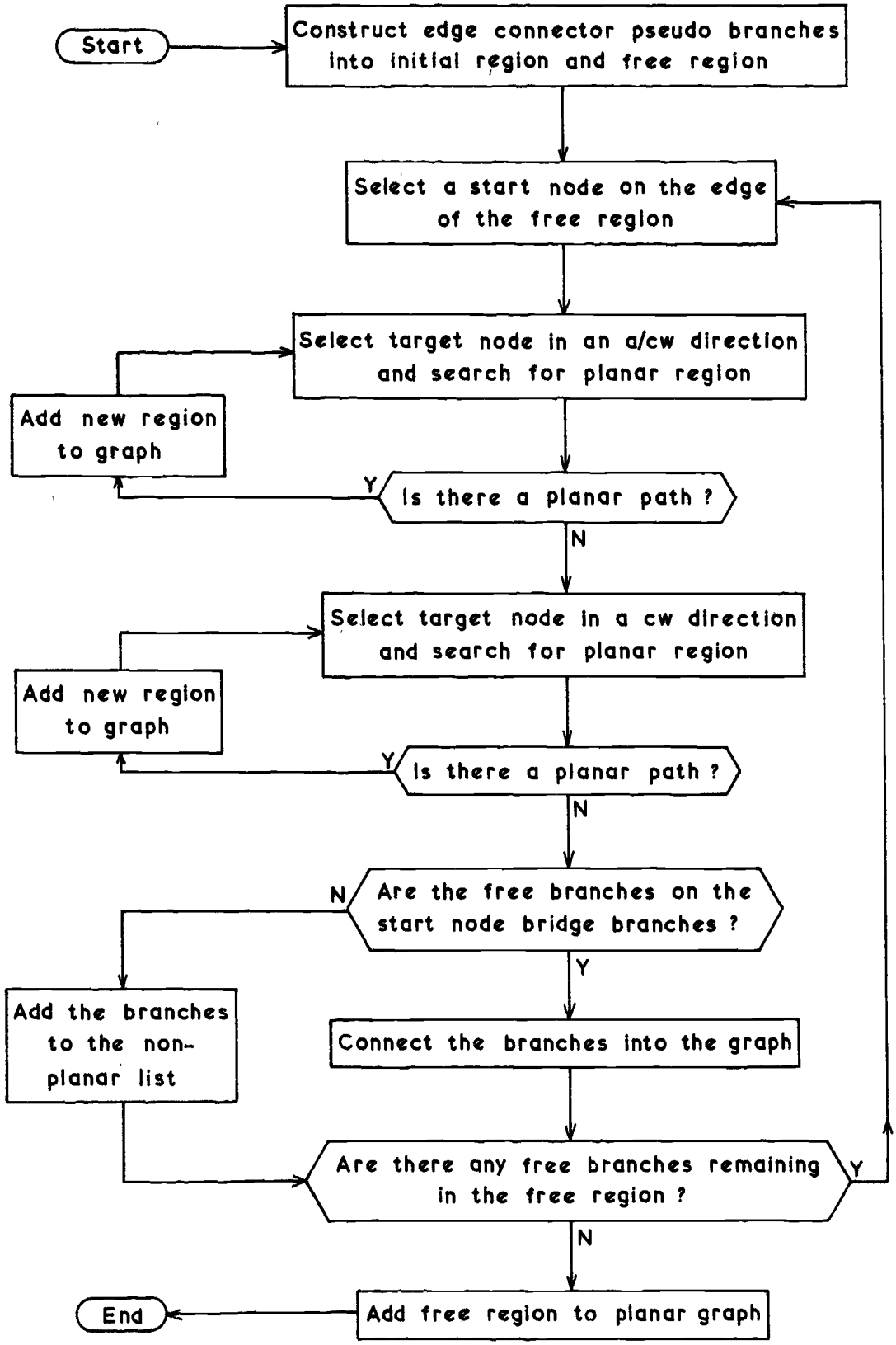


Fig. 4.5 Flow diagram of graph planarity program

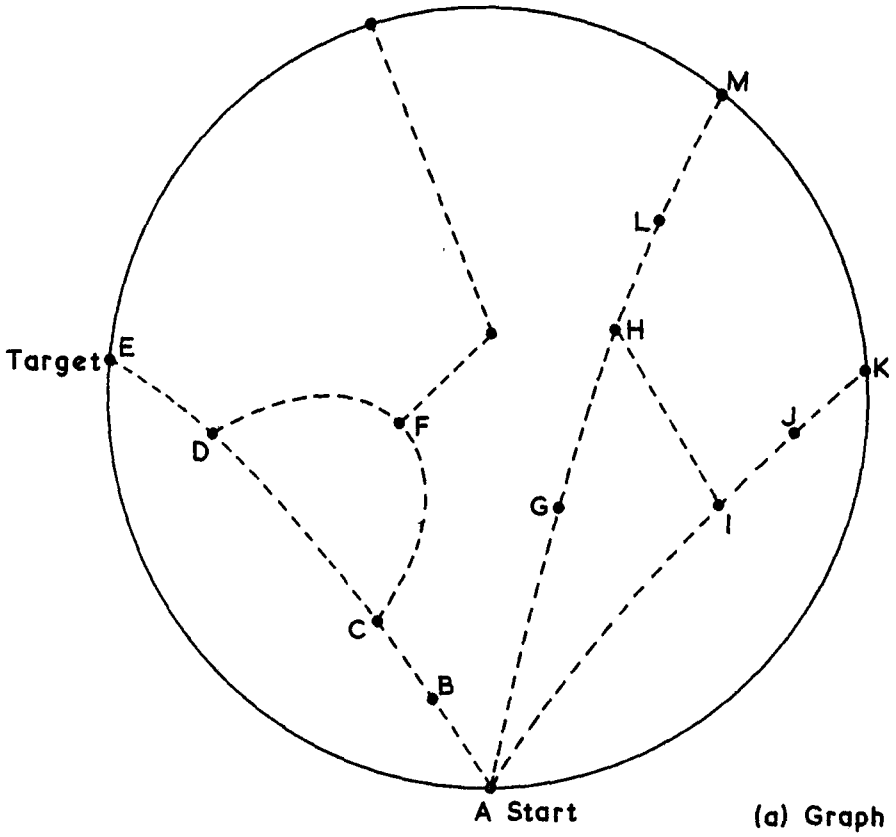
in an anticlockwise direction from the start node is taken as the target node. A search is then made for a planar path between the start and target nodes.

4.4.1 Search Procedure for Planar Paths

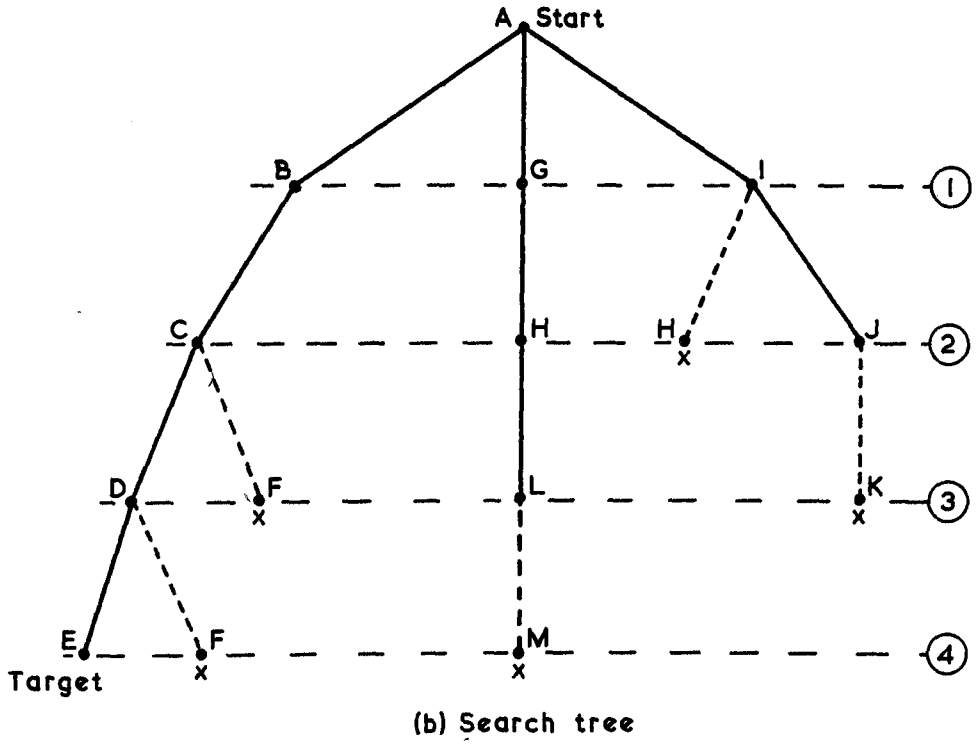
The search procedure for finding a planar path is designed to find the shortest path from start to target node. A tree search method is employed, as illustrated by Fig. 4.6. The free branches connected to the start node enable a set of the free nodes adjacent to the start node to be obtained. This set of free nodes represents the first level of the tree search. The nodes in the first level of the tree are connected by further free branches to another set of free nodes which go to make up the second level of the tree. The tree may thus be built up in successive levels until the target node is reached. All the possible nodes in one level of the tree are found before developing the next level. This ensures that the path found to the target node is of minimum length.

Each node is allowed to appear in the search tree once only. This prevents any part of the search from looping continually around a closed ring of branches. Node H in Fig. 4.6(b) for example, is reached from node G first so it is not listed as a successor to node I. No planar path is allowed to cross the free region and thus divide it into two separate regions, apart from a path between the start and target nodes. If a node on the edge of the free region is encountered during the tree search, node K for example, that part of the search is not continued.

When a subgraph component is encountered, CFDC for example, the tree search proceeds only in a clockwise direction from the



(a) Graph



(b) Search tree

Fig. 4.6 Planar path search

subgraph node, node C. The correct subgraph component orientation is thus preserved in the graph. Similarly, only one pseudo branch of the subgraph is allowed in the search path. This prevents any subgraph nodes with free branches attached from becoming embedded within a planar region. In the computer data structure representing the tree, each node is given a pointer back to its predecessor in the tree. When the target node is found, the path back to the start node may thus be directly traced. If the tree is constructed as far as is possible without reaching the target node, the branches on the start node are either non-planar or bridge branches.

4.4.2 Region Construction

When a planar path is found between the start and target nodes of a graph, the branches of the path are connected in the computer data structure as two segments of region boundaries, corresponding to the two sides of the path. The edge of the free region is divided at the start and target nodes into two separate parts. The two parts of the free region edge and two parts of the planar path are joined to form a new planar region and a redefined free region edge. If the planar path contains any subgraph nodes, the remainder of the subgraph components are also added to the graph as new planar regions.

4.4.3 Further Search Procedures

Each time a new planar region is added to the graph, a new target node is found in an anticlockwise direction from the start node. When no further planar paths can be found, the search is continued by selecting target nodes in a clockwise direction from the start node. In this case, the search for a planar path is actually made from the target to the start node so as to preserve

the correct orientation of path search.

Any branches remaining on the start node after the planar path search is exhausted are checked for non-planarity. A tree search is made from the node at the other end of the branch under consideration. If the search encounters any node on the edge of the free region, the branch is non-planar and is removed from the graph. If the search is exhausted before reaching a free region edge node, the branch is a bridge branch and so is inserted into the planar graph.

The next node with free branches in an anticlockwise direction from the start node is taken as the new start node and the search for planar regions is continued. The process is terminated when there are no free branches left in the free region. The free region itself is then added to the planar graph as the final region. The result of the planarity subroutine is thus a set of regions describing a planar graph and a list of non-planar branches. The list of non-planar branches may contain several planar branches, as the branches with which they conflicted have also been removed from the graph.

Chapter 5 Insertion of Non-Planar Branches

The planarity algorithm described in the previous chapter processes the topological representation of an electrical circuit into a planar graph and a list of non-planar branches. As these branches still represent parts of the circuit they must be included in the physical layout. An algorithm is described in this chapter for inserting these non-planar branches back into the graph.

5.1 Statement of the Problem

In the average planar graph many of its branches are either component or subgraph pseudo branches. Each of these branches may have a dimension associated with it, corresponding to the distance between two pins of its component. It is possible for a limited number of conductors to pass between two such pins, depending on the dimensions of the component and the conductors. Correspondingly, each branch in the graph may be crossed by a limited number of other branches. The crossings represent a conductor on one side of the board passing under part of a component on the other side. The condition of planarity of the topological model may thus be partially relaxed in order to allow the non-planar branches to be inserted back into the graph.

The aim of the algorithm described here is to insert all the non-planar branches of a graph into the planar subset of the graph by allowing certain types of branch crossings. The resultant graph is termed a pseudo planar graph as it may be drawn onto a plane to represent a planar set of conductor paths even though the graph contains some branch crossings. For some circuits it may not be possible to insert all of the non-planar branches into the pseudo planar graph. Two alternative procedures may then be used to deal

with these branches. The first alternative is to replace the branch by an insulated piece of wire, called a wire jumper, to make the required electrical connection. The second alternative, not considered in the scope of this project, is to route the branch as a conductor on the second side of the board.

5.2 Principles of Branch Insertion

A non-planar branch to be inserted into the pseudo planar graph may be one of two types. The first type is a component branch representing a two pin component. As the component is a part of the circuit and layout, its branch must appear in the graph. The second type of non-planar branch is a subgraph link branch. As this type represents a conductor joining a subgraph component to the rest of the circuit, it may be replaced by a wire jumper if an insertion path cannot be found in the graph. It is more important that component branches are inserted into the graph because they cannot be replaced by jumpers. They are therefore given precedence in the insertion algorithm.

A non-planar branch is inserted into the pseudo planar graph by finding a path which crosses a number of branches in the graph. The main objective is to use a minimum number of crossings when inserting each branch. This results in more room under component and pseudo branches for inserting further non-planar branches and it also helps to reduce conductor lengths in the physical layout. The list of non-planar branches to be inserted into the graph may contain several planar branches. These were originally removed from the graph because they conflicted with other branches. At a later stage all the branches with which they conflicted were also removed. When a planar branch is encountered in the list of

non-planar branches therefore, it is inserted back into the pseudo planar graph without branch crossings.

A link branch is inserted into the graph by searching for a path under component or pseudo branches from one of the branch nodes, called the start node, to the other branch node, called the target node. An example is shown in Fig. 5.1. It is assumed that branches

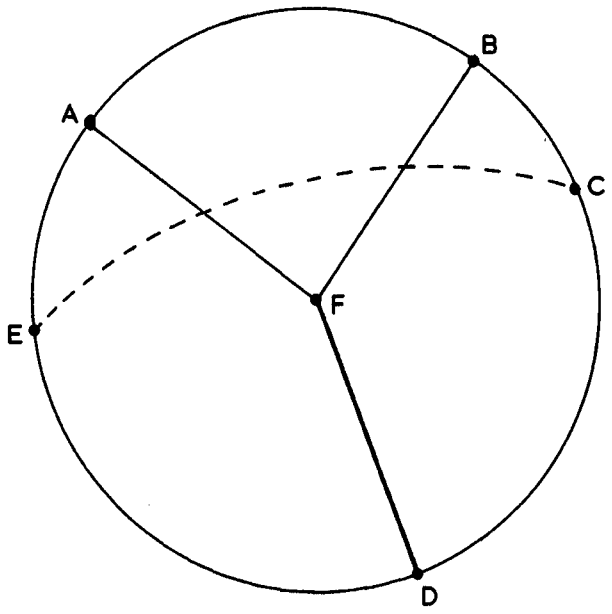


Fig. 5.1 Insertion of conductor branch

AF and BF in the figure are component branches, DF is a link branch and EC is the non-planar branch to be inserted. Branch EC is inserted by crossing under branches AF and BF. Although a shorter path exists across branch DF, two link branches or conductors cannot be crossed on a single sided printed wiring board. In a purely topological problem, EC could be routed around the outside edge of the graph without crossing any branches. In the topological representation of a board layout however, all branches must lie within the outside edge of the graph.

A different procedure is adopted for inserting non-planar component branches into the graph. A two pin component has a clearance between its pins so it is able to "hop over" several conductors on the board. A component branch may therefore be inserted into the graph by crossing over several branches representing conductor paths. The method by which this is implemented is illustrated in Fig. 5.2. The non-planar branch HD may be inserted by crossing component branches AJ, BJ and CJ as shown in Fig. 5.2(a).

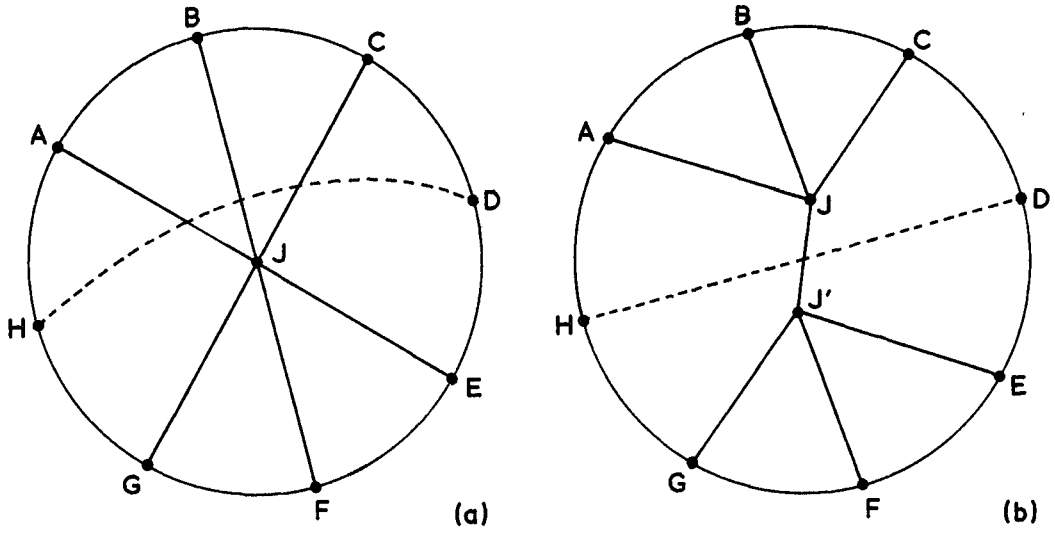


Fig. 5.2 Insertion of component branch

The number of branches over which a component branch can cross is limited by the dimensions of the component. An insertion path for the component is therefore more likely to be found if the number of crossings can be reduced. The method for reducing the number of crossings to a minimum is depicted by Fig. 5.2(b). The node J in Fig. 5.2(a) represents a point of common electrical connection of several components. The function of the electrical circuit is unchanged if the node is "split" into two separate nodes J and J' as in Fig. 5.2(b), and joined by a branch JJ' termed a conductor branch. The non-planar branch may then be inserted into the graph with a minimum number of crossings as shown.

5.3 Insertion Path Searching

The graph produced by the planarity algorithm is defined by a set of planar regions. In crossing a branch of the graph, a non-planar path passes from one region to an adjacent one. Finding an insertion path with a minimum number of branch crossings is thus equivalent to finding a path that passes through a minimum number of regions. A tree search through regions, similar to the method described in Chapter 4.4.1 is therefore used to search for an insertion path.

Each non-planar branch is initially connected to its start and target nodes in the topological representation of the circuit. As an alternative it may later be reconnected to any node which is electrically common with the start or target nodes. When starting the search for an insertion path, every region which includes the target node is marked as a target region. Similarly, regions containing nodes electrically common with the target node are also marked.

The tree search through regions is initiated by making a list of all the regions around the start node and any of its electrically common nodes. This list forms the initial level of the tree. If any of its regions have already been marked as target regions, the branch to be inserted is planar and may be inserted directly into the graph without branch crossings. At all stages of the path search for non-planar branches, the outside region of the graph is ignored as no branch may cross over the perimeter of the board.

5.3.1 Component Branch Search

To proceed with the tree search for a component branch path, a region from the initial level of the tree is examined. Each branch around the edge of the region is checked in turn. If the node at either end of the branch is a connection node and has not yet been included in the search it is further examined. A list of all the regions around the node, excluding those already in the tree, is obtained and added to the next level of the tree. These are the regions which may be accessed by splitting the node and crossing the component over the resultant conductor branch. The procedure is repeated for each region in the initial level of the tree in order to complete the list of regions in the next level.

The search procedure is repeated for successive levels of the tree. Each level is fully developed before constructing the next so that when an insertion path is found it is of minimum length. As each new region is added to the tree a check is made to see if it has been marked as a target region. The search procedure is completed when a target region is encountered. As each node is examined during the search it is given a pointer back to the region from which it was found. Similarly each region is given a pointer to the node from which it was found. This enables the required insertion path to be traced rapidly back through the tree to the start region when a target region has been found.

The number of branches that a component may cross over is limited by its physical dimensions. This in turn limits the number of levels to which the tree search may be taken. If the maximum allowable number of levels in the tree is reached before a target region is found, the component cannot be inserted into the graph by

crossing over conductors. A possible method of then inserting the component is discussed in Chapter 11.1.

5.3.2 Link Branch Search

Every component and pseudo branch has a dimension associated with it which indicates the space available for conductors to cross under the component or subgraph. This dimension is initially set during the DATAIN subroutine and may later be decremented by one conductor width each time a branch is crossed under the component. To proceed with the tree search for a link branch path, a region from the initial level of the tree is examined. Each branch around the edge of the region is checked in turn. If it is a component or pseudo branch it is further examined. If there is still sufficient clearance under the branch, the region on the other side is added to the next level of the tree. This assumes that the region is not already in the tree. The procedure is repeated for each region of the initial level in turn in order to completely develop the next level of the tree.

The tree search is continued with successive levels until a target region is reached. There is no limit to the number of branches that a link branch may cross. During the construction of the tree, each region is given a pointer back to the branch from which it was developed. This enables the required path to be traced directly through the tree when a target region is found. If the tree search is exhausted before a target region is found, the link branch is truly non-planar and cannot be inserted into the pseudo planar graph.

5.4 Path Construction

Having found the required insertion path for a non-planar branch, the pseudo planar graph has to be modified to include the branch. The initial step for the insertion of a component branch is to split all the nodes which lie along the insertion path. By using the pointers set up during the tree search, each node along the path may be identified and split into two separate nodes in turn.

The branches on a node which is to be split are divided into two groups. The groups are separated by the two regions through which the component branch is to pass. A new node is created and the branches of one group are connected to it. A conductor branch is constructed between the original and the new node and is inserted as an extra branch into the two regions. The region nearest the target node is given a pointer to the conductor branch so that the path of the component branch may still be traced. Having split the required nodes, the insertion of the component branch proceeds as for a link branch.

To insert a link branch into the graph, one end of the branch is firstly connected to the target node. The target region gives a pointer to the first branch which is to be crossed. This branch is then divided into two separate segments (The representation of branch segments is described in detail in Chapter 6.2.3.) The first segment of the link branch is also created. The segments of the two intersecting branches are linked together so that the target region is divided into two separate regions. The two regions each contain the target node and have the first link branch segment as a common boundary.

The insertion procedure is repeated along the path, dividing each region and crossed branch into two parts and creating another segment of the link branch. When the start region is reached, the second end of the link branch is connected to the start node. The start region is thus divided into two and the path is completed. Each branch segment of the graph may later be subdivided when further non-planar branches are inserted. The insertion procedure for component branches is exactly the same as for link branches. It is merely the type of branch which defines which of two crossing branches is placed on the conductor side of the board.

An example of link branch insertion is shown in Fig. 5.3.

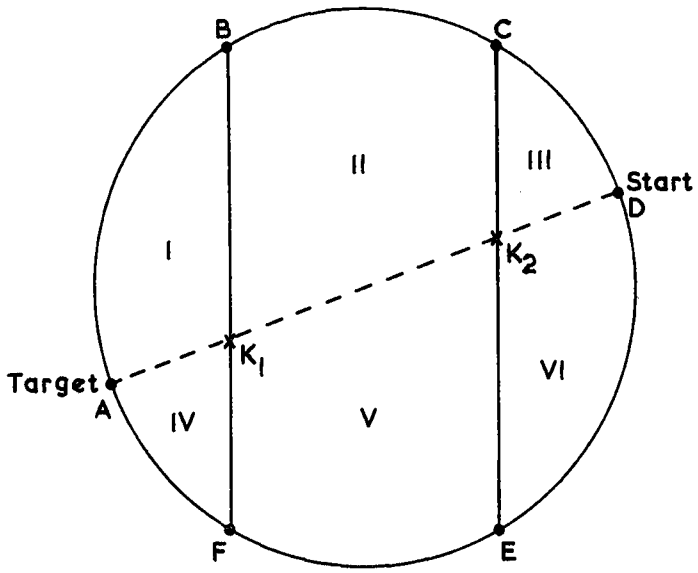


Fig. 5.3 Insertion of link branch

The three original regions of the graph,I, II and III are ABFA, BCEFB and CDEC respectively. The link branch to be inserted, AD, is connected firstly to its target node, A. Branch BF is divided into the segments BK_1 and K_1F. The first segment of the link branch, AK_1, is formed and connected to the segments of BF so that region I is divided into regions ABK_1A and AK_1FA. Region II is similarly

divided into two and region III is also divided by connecting the second end of AD to its start node. The path is thus completed with two crossings K_1 and K_2. As each component or pseudo branch is crossed, its clearance value is decremented by one conductor width.

5.5 Branch Insertion Subroutine

A subroutine to perform the above described algorithm, called PHASE2, has been written and is shown in flow diagram form by Fig. 5.4. The list of non-planar branches is initially sorted so that all component branches are in the first part of the list. The first branch is taken from the non-planar list and the graph is searched for a suitable insertion path. If a path is found the branch is inserted by the previously described methods. If the branch is found to be planar, it is inserted into the graph by connecting it across the region in which its two nodes lie. The region is thus divided into two separate regions.

Any branch for which no path can be found is put into a second list of non-planar branches. These branches are truly non-planar and cannot be inserted into the pseudo planar graph. Non-planar link branches are later replaced by wire jumpers. Non-planar component branches may later be connected by one node into the graph, the connection to the other node being made by a wire jumper.

Having processed one branch, the procedure is repeated with the remaining branches from the non-planar list in turn until the list is exhausted. The end result of the insertion subroutine is then a pseudo planar graph which is the complete topological model of a circuit layout. There may also be a list of non-planar connections that have to be replaced by wire jumpers.

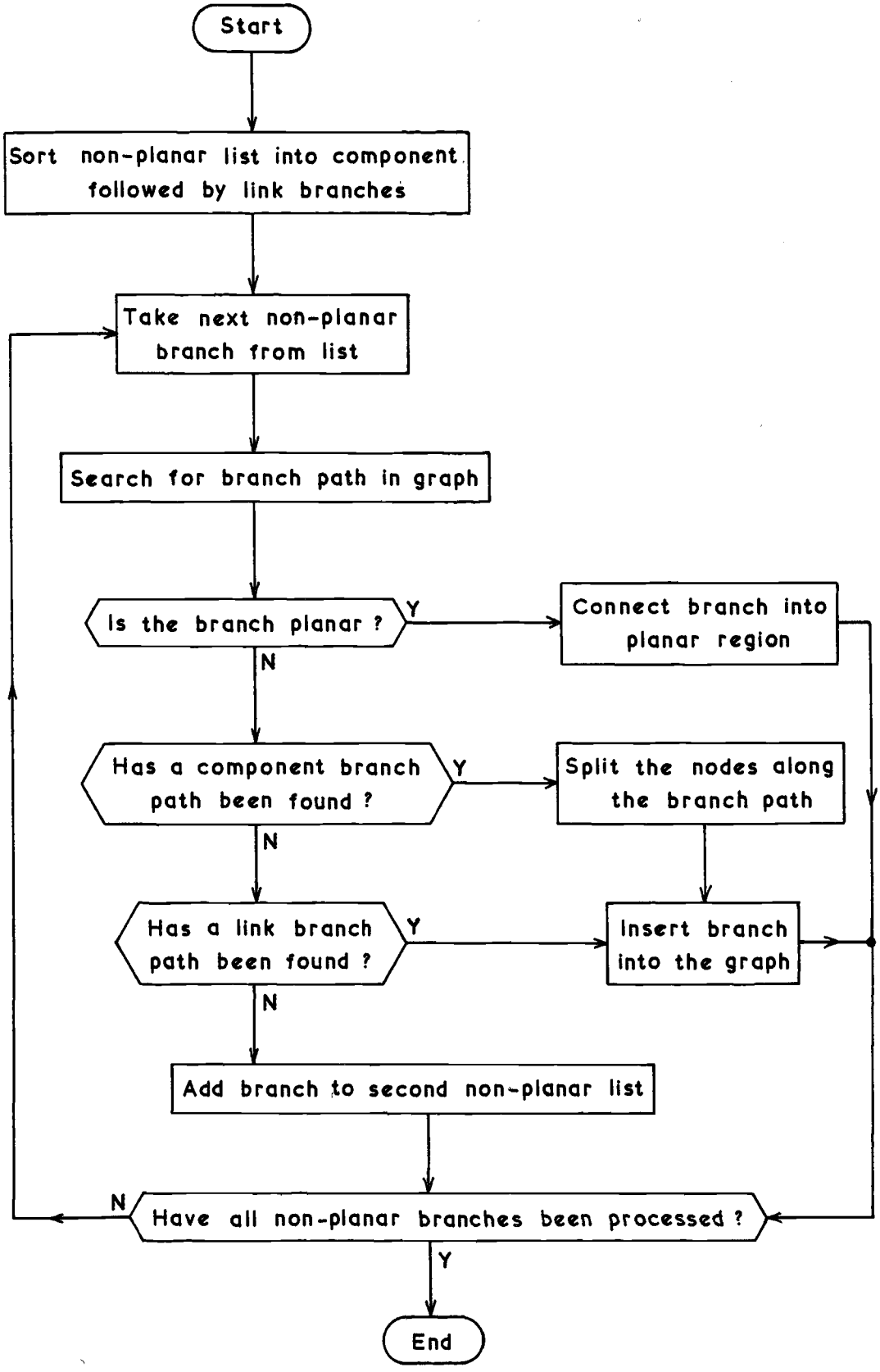


Fig. 5.4 Flow diagram of PHASE2 subroutine

Chapter 6 Computer Implementation of Topology Algorithms

The algorithms for constructing the topological model of a circuit have been described in the previous chapters. In this chapter the programming methods used to implement the algorithms are described. The computer hardware configuration has already been outlined in Chapter 1.

6.1 Data Storage

The representation of an electrical circuit consists of a large number of interconnected nodes, branches, subgraphs and planar regions. In addition, the branches and subgraphs representing components require component names and, at a later stage, physical co-ordinates. The problem is to devise a system to store all of this information in a compact and readily accessible form.

A data storage system similar to that described by Ross (27) is used. A large one-dimensional array is assigned as a common area in which to store all the data. The area within this array is divided up into a large number of blocks. Each block consists of a number of consecutive elements of the array and may be of any length. A block is used to represent a node, a branch or any other element of the graph. Interconnections of the blocks are represented by pointers. A pointer to a block is merely the array index of the first element of the block.

A free storage system is used to allocate blocks from the array for use by the various subroutines. During the topological algorithms it so happens that no block ever becomes redundant. An elementary free storage system is therefore used although a more complex one is described in Chapter 9.2 for use with the layout

algorithms. The storage system uses a pointer, set initially to the beginning of the array, to indicate the start of the un-used part of the array. When a new block is required, it is taken from the free part of the array and the storage pointer is incremented by the corresponding block length. After each block has been allocated, the value of the storage pointer is checked to ensure that the limit of the array has not been exceeded.

6.2 Data Structure

The graph of a typical circuit contains many hundreds of interconnected blocks. It is important therefore to use a data structure which is efficient in describing the interconnections. There are a number of general purpose data structure packages available, such as ASP (17, 25), which may be used with FORTRAN programs. Being general purpose packages however, they tend to have large overheads in storage space when defining block interconnections. A special purpose data structure has therefore been designed for use with the planarity and layout algorithms described here. It is organised with interrelated parts of the graph closely connected by pointers so that one may move easily from one part of the structure to another.

A general purpose data structure package usually contains checks to ensure that each operation on a block is a valid one. The disadvantages of this are that extra storage space is required in each block to indicate its type and that the program requires extra execution time for each operation to be checked. The data structure developed here has no such checks and so saves on storage space and computing time. The disadvantage is that the program generally fails completely if an invalid operation is performed.

6.2.1 Interconnection of Nodes and Branches

The method of interconnecting nodes and branches in the data structure is illustrated by Fig. 6.1. A simple graph is shown in Fig. 6.1(a) and its resultant data structure is shown in Fig. 6.1(b). The first element of each node block contains the name of the node, each node having a unique name. The nodes of a circuit are all held in a node list. The second element in each node block is thus used to point to the next node in the list. The list is terminated in the final node block by a zero value pointer. Each node has a number of branches connected to it. The third element in a node block thus points to the first branch which is connected to it. The remainder of each node block is used as a workspace in which to store various markers and pointers during the course of computation.

The first element in a branch block is a marker describing the type of the branch, for example a component, or pseudo, or link branch. The next two elements of the branch point to the two node blocks between which the branch is connected. The two following elements of the branch block are used to form the list of branches connected to a node. The first of the elements corresponds to the first node pointer and the second element to the second node pointer. Each of the list elements points to the next branch connected to the node, or has zero value for the last branch in the list. The remaining elements of the block are used for workspace and for connections to other parts of the data structure which are described later.

As an example of the type of operation required on the data structure, all the branches connected to node 2 in Fig. 6.1 are to be found. The branch pointer in the node block N2 points to branch

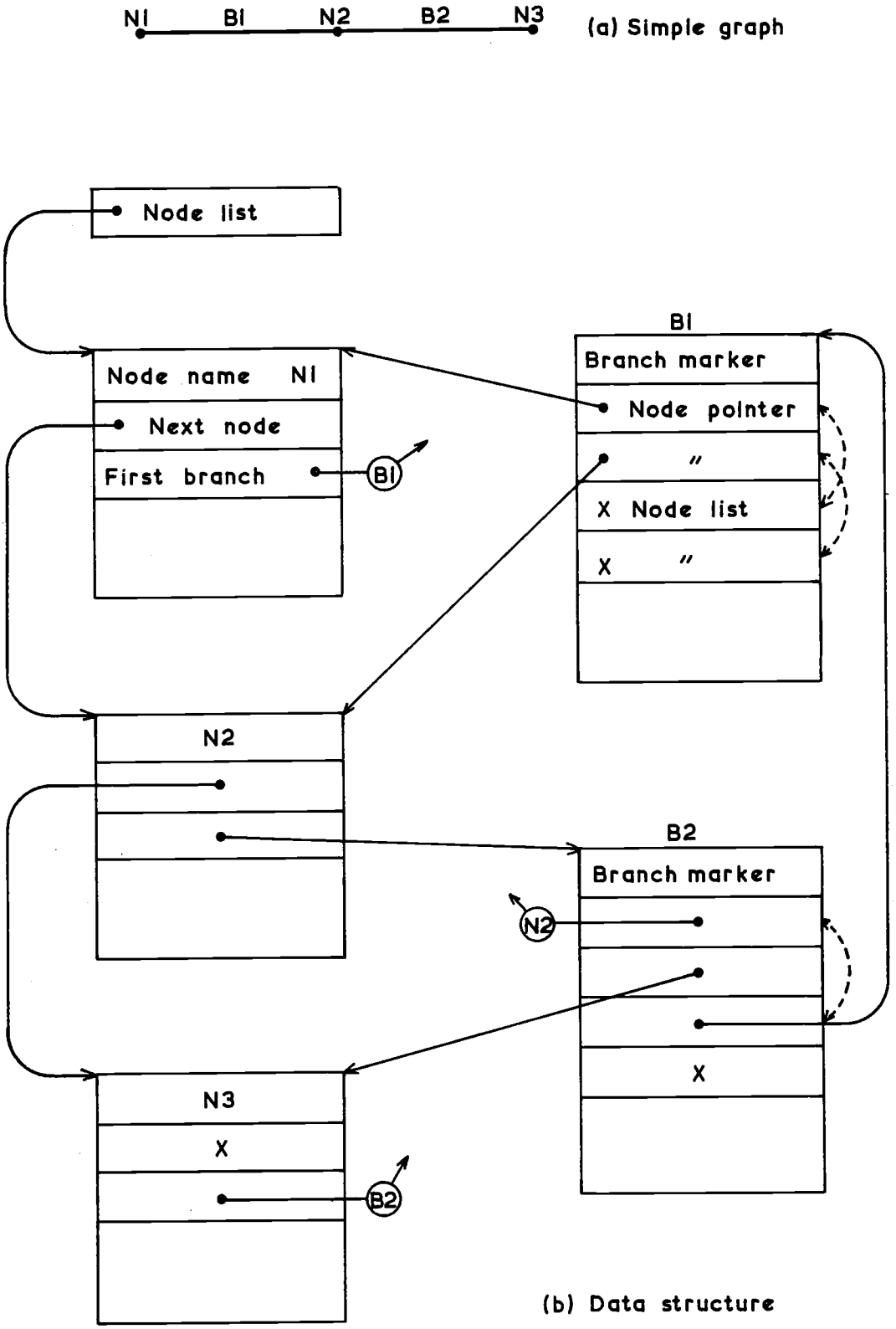


Fig. 6.1 Interconnection of nodes and branches

block B2 which is thus the first connected branch. As the first node pointer of B2 points to N2, the corresponding first node list pointer of B2 is taken. This points to branch B1 which is thus the second branch connected to N2. In this case node 2 is the second node of the branch so the second node list pointer of B1 is taken. The pointer is a null one which indicates the end of the list so B1 and B2 are the only branches connected to the node. The pointers are arranged in this way so that new branches may be added to a node without having to alter the length of its node block. The method of interconnection enables one to readily find all the branches connected to a node and vice versa.

6.2.2 Subgraph and Branch Components

The method of defining the constituent parts of a subgraph component is illustrated by Fig. 6.2. Parts of the structure have been omitted from the diagram to avoid confusion. The nodes and branches of the subgraph are interconnected in the same manner as described in the previous section. This ensures that the parts of the subgraph are compatible with the rest of the graph when constructing planar regions.

The overall component is described by a subgraph block, S1. The first element of the block is a subgraph marker. The second element is a pointer to the first subgraph node of the component, SN1. The subgraph nodes are held in a list, in the same way as circuit nodes. The difference is that the start of the list is stored in the subgraph block and the last node has a pointer back to the subgraph block. The subgraph nodes are thus joined in a ring together with the subgraph block. In addition, the first element of each node block has a marker plus a pointer back to the subgraph

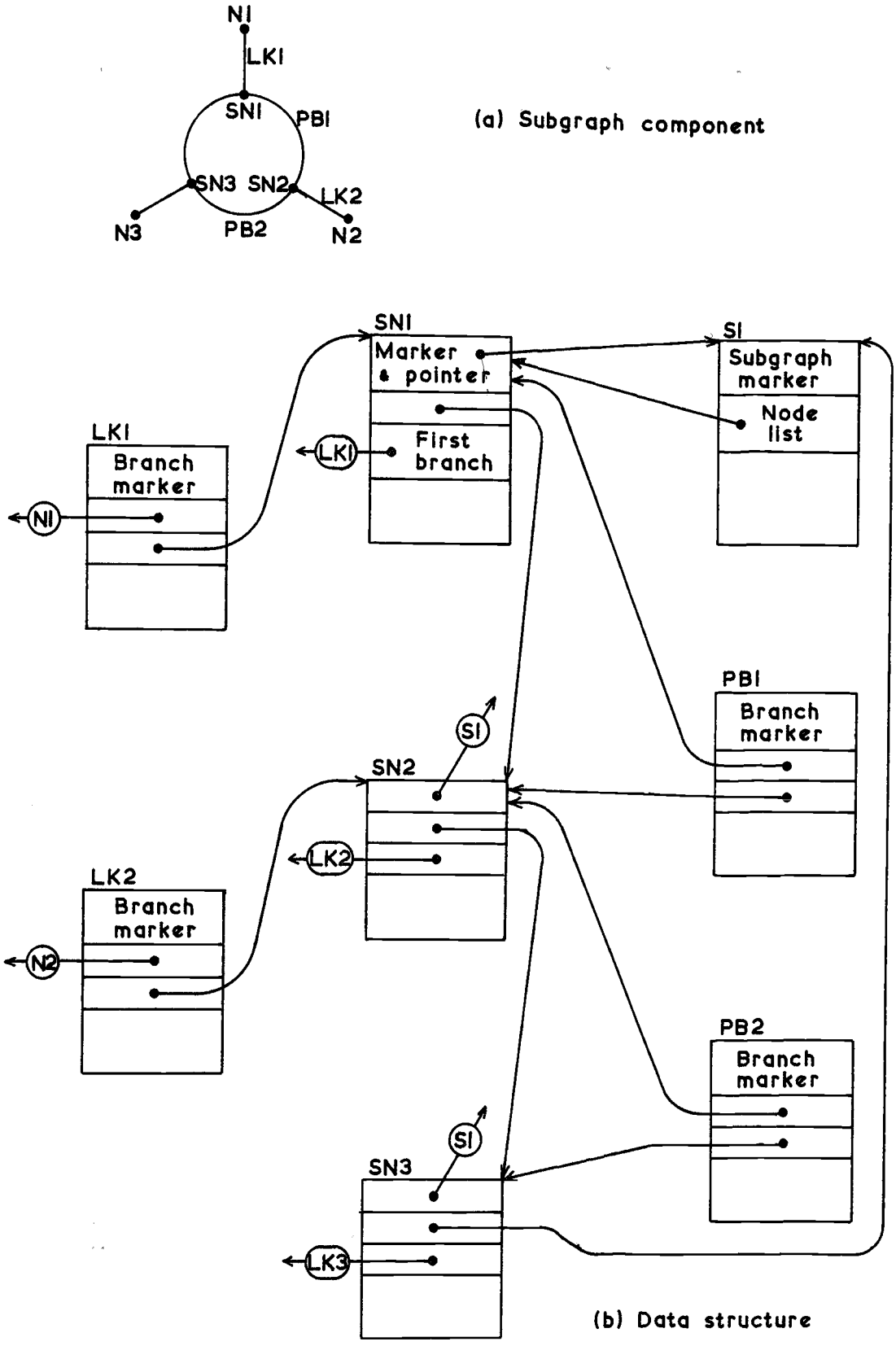
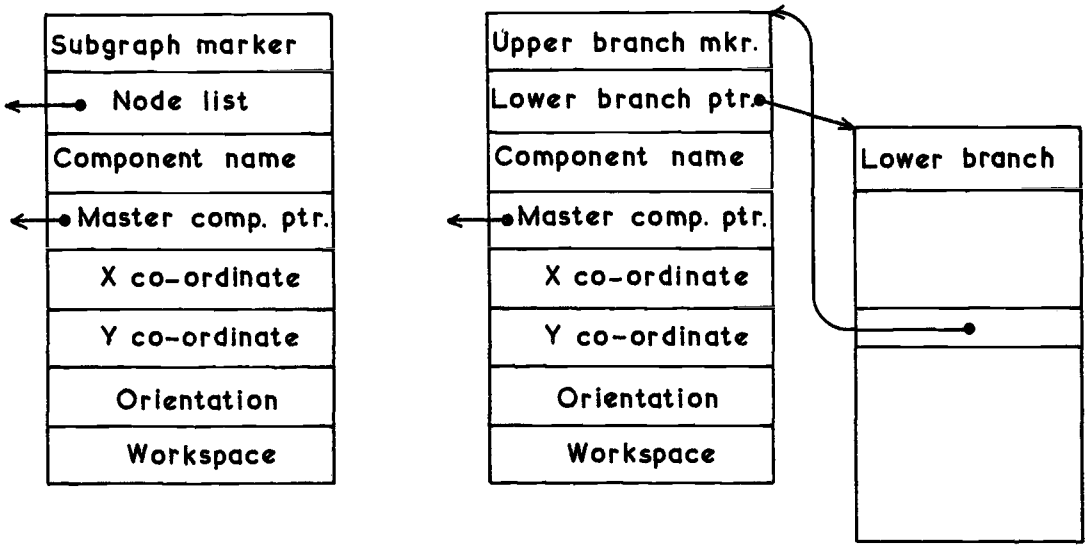


Fig. 6.2 Interconnection of subgraph nodes and branches

block. Given a subgraph node therefore, one may readily find the subgraph component to which it belongs.

Every subgraph node has a list of the branches attached to it; these are a link branch and two pseudo branches. The pseudo branches each have two pointers to their appropriate subgraph nodes. The link branch also has two node pointers. The first pointer identifies a circuit node and the second points to the corresponding subgraph node. The structure of a subgraph component is thus defined completely in a manner which is compatible with the remainder of the graph.

At a later stage of the layout algorithm when components are given physical co-ordinates, it is desirable that both the branch and the subgraph component blocks are compatible. The form of a subgraph block is shown in Fig. 6.3(a). This is the same block as the one



(a) Subgraph component block (b) Branch component block

Fig. 6.3 Component data blocks

marked S1 in Fig. 6.2(b). The third element of the block contains the characters of the user name of the particular component, for example TR1 or TR2. The fourth element is a pointer to a master component block in the component library. The master block saves repetition of information common to every component of a particular type; its contents are discussed in Chapter 3.3.1. Further elements in the subgraph block store the physical co-ordinates and orientation of the component and provide working space for the layout algorithm.

The data structure for a branch component should be compatible with the subgraph block just described. It should also be compatible with the method of interconnecting nodes and branches described in the previous section. These two requirements both use the same area of a block and so are mutually exclusive within the same block. The data for a branch component is therefore divided between two blocks as shown in Fig. 6.3(b). The upper block is identical to the subgraph block apart from the first two elements. The first element contains a marker describing the type of block; the second contains a pointer to the lower branch block. The lower block describes the interconnections of the branch into the graph and corresponds to either of the blocks marked B1 or B2 in Fig. 6.1(b). It also has a pointer back to the upper branch block.

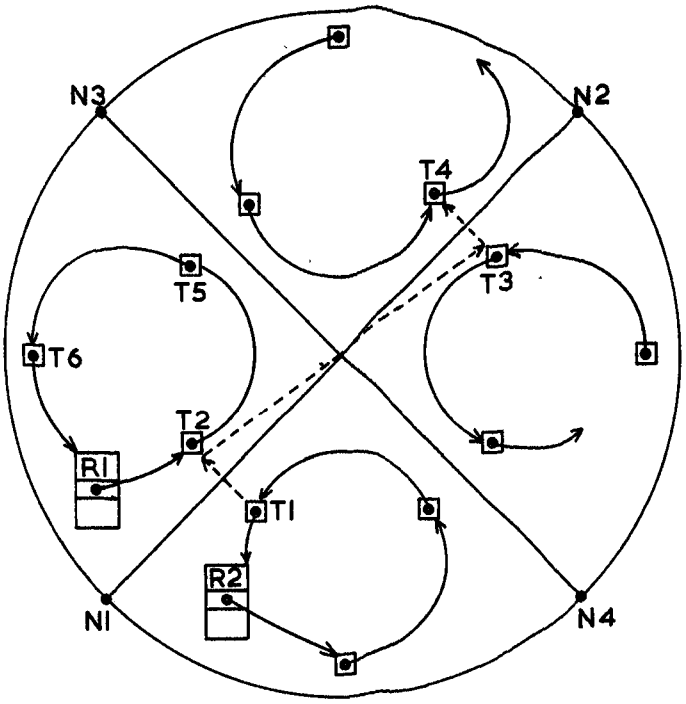
6.2.3 Branch Segments and Planar Regions

The pseudo planar graph of a circuit is defined by a set of planar regions. Certain branches of the graph may each be divided into a number of branch segments by other crossing branches, in the manner described in Chapter 5.4. In developing a data structure to represent this type of graph, two problems arise. The first is to

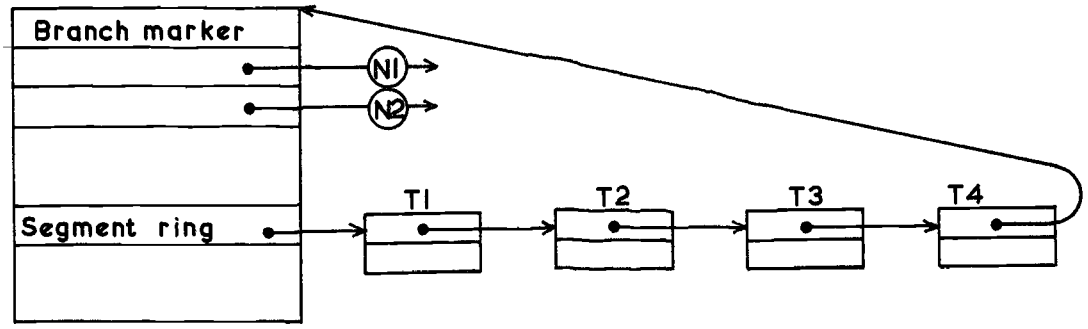
define the correct sequence of branch segments around the edge of a region. This sequence describes the order in which branches are connected so as to avoid branch intersections in the planar graph. The correct sequence is also necessary at the layout stage so as to give the correct order of component connections. The second problem is to define the correct sequence of segments from one end of a branch to the other. This is essential in preventing the conductors under a component from intersecting each other.

A pseudo planar graph and its method of representation are illustrated by Fig. 6.4. In this example two of the branches, N1 to N2 and N3 to N4, intersect and divide each other into two segments as shown in Fig. 6.4(a). The linking between branches and their associated planar regions is performed by two-element blocks called tie blocks. The interconnections between branch and tie blocks are shown in Fig. 6.4(b). Every branch segment is defined by a pair of tie blocks, one for each region adjacent to the segment. If a branch is divided into several segments, it is defined by a list of tie block pairs. The order of tie block pairs corresponds to the order of segments on the branch. An element in the branch block contains a pointer to the first tie block. The first element in the tie block points to the next tie block and so on. The final tie block then points back to the branch block so that given a tie block, its corresponding branch may be found.

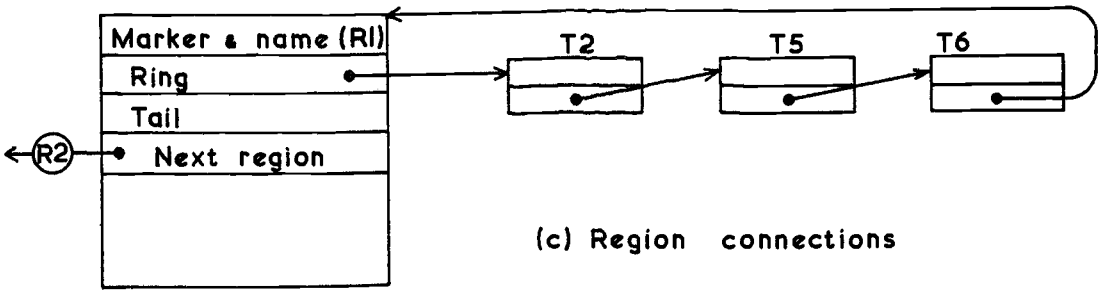
All the planar regions of a graph are represented by a list of region blocks in the data structure. The description of a planar region is illustrated in Fig. 6.4(c). The first element of a region block contains a region marker plus a unique name for the region. The second element points to the first tie block of the region.



(a) Pseudo planar graph



(b) Branch connections



(c) Region connections

Fig. 6.4 Interconnection of branches and regions

The tie blocks are connected in a ring together with the region block by the second element of each block pointing to the next tie block. The final block then points back to the region block. Each tie block in the ring belongs to a different branch segment such that the order of blocks corresponds to the order of branch segments around the planar region. The third element of the region block contains a pointer to the last tie block of the ring so that the two end blocks of the ring may be readily identified. The fourth element of the region block points to the next block in the list of regions.

The correct ordering of the segments of a branch is maintained by adopting a convention of interconnection ordering. Referring back to Fig. 6.4(a), it can be seen that all the regions are connected in an anticlockwise direction. The position of the region block within its ring of tie blocks is not important. If N1 is the first node of branch N1 to N2, it can be seen that the tie blocks on one side of the branch point towards N1 whilst those on the other side point towards N2. The tie blocks are therefore arranged on the branch ring so that the first one of every pair points towards the first node, N1, whilst the second points away from N1. In addition, the first tie block in the branch ring belongs to the branch segment nearest to the first node. The dotted lines in Fig. 6.4(a) show the order in which the tie blocks are attached to the branch. The convention of ordering thus enables the branch segments to be kept in the correct order.

6.3 Computer Language

The board layout program involves a great deal of data structure processing. One requirement of the program is that it

should be readily transferable from one computer installation to another. There is no widely used data structure processing language so it was decided to use FORTRAN IV (10) together with a general purpose macro processor ML/1 (4), for the layout program. The ICL 4130 described in Chapter 1.4 has a magnetic tape based FORTRAN system which enables programs to be compiled and run from magnetic tapes. It also enables precompiled subroutines to be stored on magnetic tape which is a useful feature when developing a large program.

The general purpose macro processor is used for the implementation of the data structure within the FORTRAN language. Some of its facilities are described in Appendix A. Statements describing operations on the data structure are written as macro calls. When completed, the program is processed by the macro processor so that all the user-defined statements, or macro calls, are replaced by FORTRAN statements. The program may then be compiled and run as a normal FORTRAN program.

An example of the use of the macro processor is described here. It is assumed that a variable, PTR, contains the index in the one-dimensional data array, IRAY, of a subgraph component block. The fifth and sixth elements of this block contain the X and Y co-ordinates of the component. The co-ordinates of the component may be obtained by using the macro calls:

```
X = COMPX(PTR)
and  Y = COMPY(PTR)
```

The definitions of the macro calls describe the replacement text for the calls so that after processing they are replaced by their equivalent FORTRAN statements, i.e.

```
X = IRAY(PTR+4)
```

and `Y = IRAY(PTR+5)`.

The above example could be implemented by the use of a FORTRAN function statement. The reason for using the macro processor is that data structure statements also need to appear on the left hand side of an assignment, for example:

```
COMPX(PTR) = X
```

This type of statement cannot be implemented by a FORTRAN function, hence the use of the macro processor.

It is clear that a program written with macro calls is far easier to understand than its equivalent FORTRAN text. Changing the order of elements in a block or changing the length of a block during the development of the program is also facilitated. Only the macro definitions need to be altered as the macro processor will automatically apply the alterations to the program during processing.

Chapter 7 Placement and Routing of Board Layout

The algorithms for automatically constructing a board layout from the topological model of a circuit are described in this chapter. A later chapter describes the modifications necessary to allow graphical interaction with the layout program.

7.1 Consideration of Layout Methods

The majority of methods for generating printed wiring board layouts split the problem into two separate stages. Component positions are computed first and the components are fixed at their appropriate co-ordinates. The conductor routing stage then becomes a problem of finding paths to connect together sets of fixed-position pins in the required order. This approach conveniently allows one to divide the layout algorithms into two lesser independent problems. The main disadvantage is that components are placed with little or no regard to the subsequent routing of conductors. If the components could later be repositioned in congested areas of the board, some further conductors might be routed where there was otherwise insufficient space between components. A further disadvantage is that considerable computing time may be wasted in searching for conductor paths that are topologically impossible to route.

The advantage of constructing a topological model initially is that the relative positions of all components and conductors are known before layout commences. This means that components can always be placed so as to allow sufficient clearance for intervening conductors. In addition, conductor routes can be constructed in steps from one component pin to the next rather than having to search for a path over a large area of the board.

7.1.1 Objectives of Board Layout Method

During the generation of a board layout, a number of objectives have to be considered. The following objectives are true whether the layout is developed from a topological model or by any other method:

(a) All the circuit components and conductor paths must be placed within the available board space. This may present some difficulty when a board is densely populated with components. In addition to the board area required by the components themselves, further space is required between them for routing the conductors.

(b) Every conductor should be of minimum length. For high frequency circuits this reduces the effects of stray capacitance upon the performance of the circuits. For all layouts, minimum length conductors reduce the amount of board space required for routing and enable more compact layouts to be generated.

(c) The spacing between adjacent components and between adjacent conductors must be greater than certain specified minimum values. Clearances between adjacent components are necessary in order to allow for such things as tolerances in component positioning, insulation between the components and heat dissipation of some components. A minimum value of spacing between the centre lines of parallel conductors must be specified to allow for the width of conductors, insulation space between conductors and manufacturing tolerances in the production of printed wiring boards.

(d) As well as the essential conditions described above, there are often a number of constraints which are peculiar to each particular layout. For example, the adjustment screw of a

potentiometer should face towards the edge of the board, or the input and output connections of a high gain amplifier should be kept apart so as to reduce the effects of inductive and capacitive coupling.

7.1.2 Force-Field Method of Layout Construction

A method of board layout studied initially for this project made use of a mechanical force analogue similar to the ACCEL program described in Chapter 2. Using the topological model of a circuit, components and conductors were initially placed so that no conductor paths intersected. The object was then to alter the placements so as to give a compact layout whilst preserving planarity. Each conductor was considered to exert a force, proportional to its length, upon its two attached components. The purpose of this force was to bring closely connected components together and reduce conductor lengths. Each component exerted a force of repulsion, inversely proportional to distance, on all adjacent components. The force was used to prevent adjacent components from over-lapping.

The conductors were each divided into a number of segments. Forces of attraction and repulsion were similarly exerted between adjacent conductor segments so as to reduce the length of each conductor, without allowing it to cross any other conductor. The algorithm proceeded in an iterative manner, moving every component and conductor segment a distance proportional to the net force upon it. The algorithm terminated when all the components and conductors reached stable positions within the available board space.

A program was written to make a simplified study of the problem. The main drawback encountered was that each conductor had

to be divided into many segments to ensure that no parts of adjacent conductors crossed. The problems associated with the storage of large amounts of data, together with the time required to perform many iterations of the program, made this approach unsuitable for large board layouts. The results of the program were also critically dependent on the relative values of attraction and repulsion forces. An inductive method of constructing board layouts was therefore developed, described in the following sections.

7.2 Principle of Layout Algorithm

The method used to construct the board layout of a circuit from its topological model builds up the layout in a logical series of steps from a known starting point. The type of board considered is a rectangular board with an edge connector along one side. A list of all the components connected to the edge connector can be obtained from the topological model. These components may be placed in a strip across the width of the board, parallel and adjacent to the edge connector. The topological model then gives a list of the components connected to those already on the board. The layout may thus be constructed by placing components in a series of parallel strips across the board, working from the edge connector to the opposite side of the board.

At any time during the layout construction, a boundary line may be drawn across the board separating the part of the board occupied by components from the unoccupied part. This is illustrated by the dotted line shown in Fig. 7.1. The boundary of the unoccupied part of the board is thus divided into a number of slots. The lower edge of each slot is coincident with the upper side of a placed component. The two sides of each slot are coincident with either the

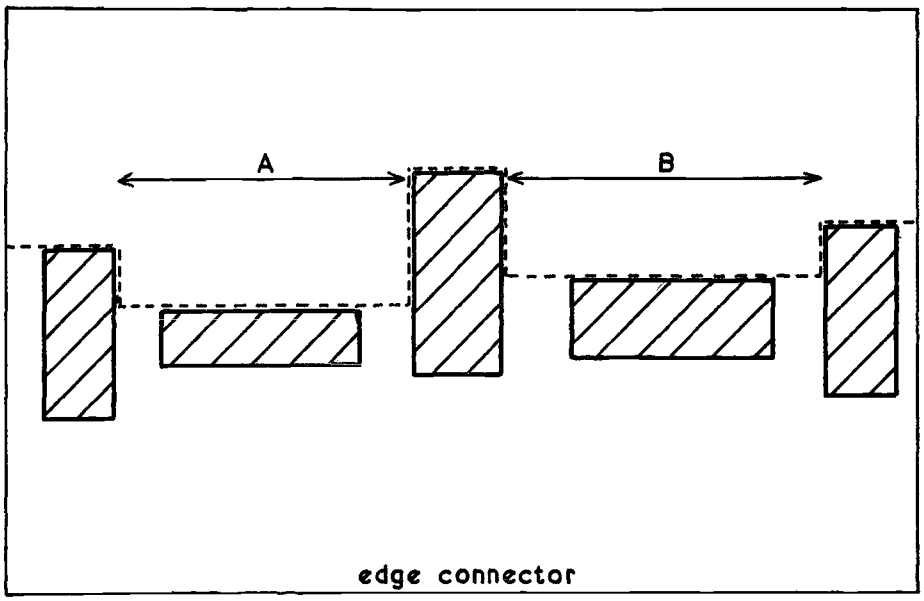


Fig. 7.1 Board layout slots

sides of components or the sides of the board. The width of the two slots shown in Fig. 7.1 are indicated by the measurements A and B.

The board layout is constructed by placing components into successive slots of the unoccupied part of the board. This is in preference to using parallel strips across the whole width of the board due to the irregular shape of the placed component boundary. The initial slot of the layout is coincident with the lower edge and two sides of the board. Thereafter, the next slot chosen for component placement is the lowest slot (the one nearest to the edge connector), working from left to right across the board. As components are placed in a slot, the boundary of the unoccupied part of the board is updated, thus creating new slots.

The processing of a slot is performed in two stages. The first stage consists of node development and sorting. Around the edges of the slot are conductors, or circuit nodes, from the

occupied part of the board. Reference is made back to the topological model to obtain a list of all the components and conductors connected to these nodes. The list is then sorted to determine the optimum set of components to place in the slot. The second stage of slot development consists of component placement and conductor routing. The physical co-ordinates and orientations of the components are calculated. The conductors are then routed from the edges of the slots to the appropriate component pins.

7.2.1 Aims of Layout Algorithm

When placing components into a slot, the main objective is to pack in as many components and conductors as possible. Before construction of the layout commences, there is little indication of the final component and conductor density on the board. Slots are therefore closely packed to ensure that the layout will fit onto the board. If the board is not densely populated with components there will be a large strip of unoccupied space across its upper width when the layout is completed. The component and conductor co-ordinates may readily be multiplied by a scale factor in the Y direction so as to occupy the whole board if desired.

The method of constructing the board layout is also aimed at producing minimal conductor lengths. In deciding the contents of a slot, the components chosen are those most closely connected to the existing part of the layout. The interconnecting conductors between components thus tend to be of minimal length. The correct clearances between adjacent components and conductors are also to be maintained by the layout algorithm. They may readily be computed to their correct values because the layout is constructed in a series of successive slots. Special constraints such as those mentioned in

section 7.1.1(d) are difficult to program for a general purpose layout algorithm. They are therefore dealt with by the interactive methods described in Chapter 8.

7.3 Slot Development and Sorting

This section describes the processing required in order to choose the optimum set of components and conductors to be placed in one slot during the construction of a layout. The slot is initially assumed to be empty and along its lower edge are the ends of a number of uncompleted conductor paths. These paths come from component pins or parts of conductor paths which have already been placed on the board at a lower level. An example is shown in Fig. 7.2;

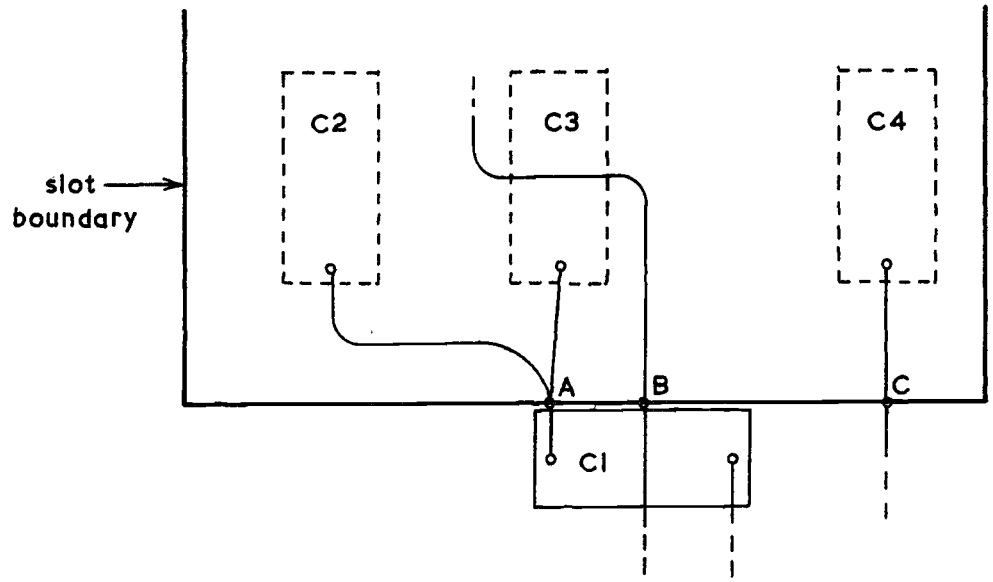


Fig. 7.2 Development of slot nodes

the component C1 has already been placed in the layout. The three conductor paths from a lower level are initially routed up to the points marked A, B and C on the lower slot boundary.

The conductor paths in the layout may be of two different types and it is appropriate here to describe the difference between them. The first type of path is the physical representation of a circuit node in the topological model. It directly connects together two or more component pins without crossing under any other components. Two examples are shown by conductor paths A and C in Fig. 7.2. The second type of path corresponds to a branch segment in the topological model. The branch concerned may be either a subgraph link branch or a conductor branch. The path is one which, when routed further in the layout, crosses under a component. An example is given by path B in Fig. 7.2. These two types of conductor paths are termed nodes and conductors respectively.

The data for processing the contents of a slot is stored in blocks, similar to those described in Chapter 6.1. The blocks are organised into two lists called the base list and the working list. The base list contains information on all of the uncompleted conductor paths at the lower edge of the slot. The working list is used for storing and processing information on all of the possible contents of the slot.

There are four different types of block which may be used in the base and working lists. These are:

(a) Branch block which holds data related to a branch component.

(b) Subgraph block which holds data related to a subgraph component

(c) Node block which relates a conductor path to all or part of a circuit node.

(d) Conductor block which relates a conductor path to a branch segment in the topological model.

Each block contains a pointer back to an appropriate part of the

topological model so as to identify the physical layout with the topological model.

7.3.1 Development of Nodes and Conductors

The initial step in finding the optimum contents of a slot involves the development of all the nodes and conductors along the lower edge of the slot. These elements are stored in the correct physical order in the base list of the slot. The development of a node or conductor is defined as creating a list of all the possible components and conductors which may be connected to the element. This list then forms part of the working list of the slot. Examples of development are shown in Fig. 7.2. Node A develops into components C2 and C3, conductor B develops into a further conductor and node C develops into component C4.

One difficulty in describing a node in the physical layout is that several parts of the same node may appear in different parts of the layout. A part of a node is defined as a conductor path connected to one or more components of a given circuit node. An example of two parts of a node in the same slot is given in Fig. 7.3. The topological model of the node and its connected components are shown in Fig. 7.3(a) whilst a possible physical representation is shown in Fig. 7.3(b). It is essential to uniquely identify each part of a node so that the parts may be developed into the correct sequence of components and conductors.

Each part of a node is uniquely identified by pointers to three different elements in the topological model. These are the corresponding circuit node and two bound branches which are connected to it. The order of branches around a node is defined in the

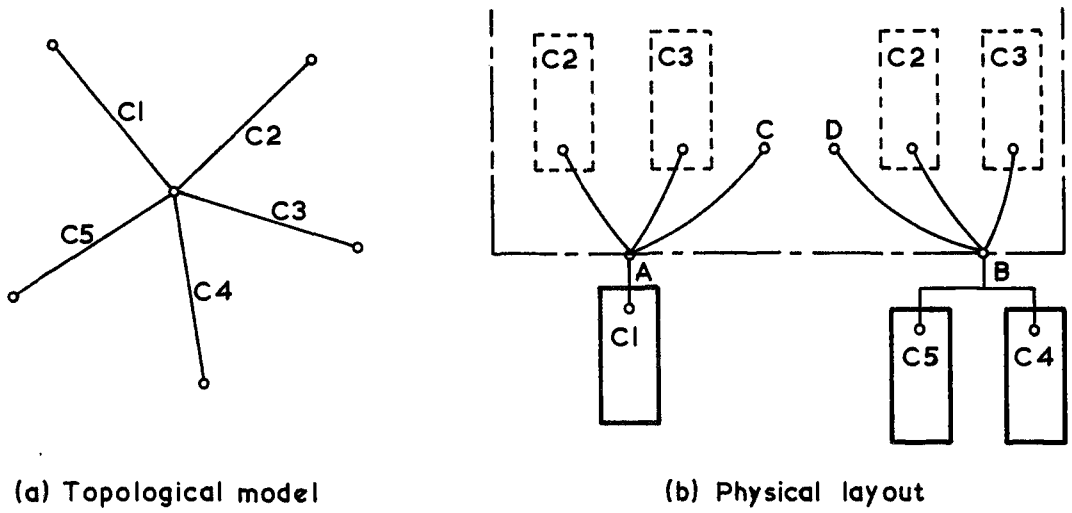


Fig. 7.3 Development of a node

topological model so there is a corresponding order of components and conductors in the physical layout. The two bound branches are defined as the first and the last branches connected to that part of the node which is already placed in the layout. The remaining branches connected to the node part, if any, are thus defined as those which lie between the two bound branches. Two examples of parts of nodes are shown in Fig. 7.3(b). The bound branches of part A are both the component C1. The first and second bound branches of part B are the components C5 and C4 respectively.

The development of a base node, or node in the base list of the slot, proceeds in a clockwise order of branches around the node. The first branch to be developed is the one following the first bound branch. If the developed branch is a component branch, the corresponding block in the topological model is checked. If the component has not yet been placed in the layout a branch block is added to the end of the working list. Otherwise, a node block is added to the working list. This represents a part of the node

which will be routed as a conductor path until it connects with the already - placed component. An example of node development is illustrated by node A in Fig. 7.3(b). Following the first bound branch C1, the components C2 and C3 are developed. The remaining components, C4 and C5, have already been placed so node C is added to the working list.

If the developed branch is a link branch, the topological model is checked to see whether it crosses under any other branch. If there are branch crossings, a conductor block is added to the working list with a pointer to the appropriate segment of the link branch. If there are no crossings the link branch must be connected directly to its subgraph component. The corresponding component block is therefore checked as before to decide whether to add a subgraph block or a node block to the working list.

The developed branch may be a conductor branch (produced by splitting a node during the construction of the pseudo planar graph). In this case there will always be a branch crossing so a conductor block is added to the end of the working list. The developed branch may also be a pseudo branch belonging to the edge connector. This means that the base node is part of an edge connector node. In this case a node block is added to the working list. The node will be routed as a conductor path until it is joined to another part of the same node which is already connected to the edge pin. The development of the base node is continued with each branch in turn, in a clockwise order around the node, until the second bound branch is encountered.

A conductor block in the base list may be developed in a similar manner to a base node. Each conductor block contains a pointer to a tie block in the topological model. The conductor path

in the layout may thus be identified with a particular branch segment and the direction in which the branch is being traversed. To develop a conductor block, the topological model is checked to find the element which follows the current branch segment. This element may be either a branch crossing or a node. In the case of a branch crossing, a duplicate conductor block is added to the end of the working list. The block is given an additional pointer to the component which it is to cross. This indicates the destination of the conductor and is used in a later part of the algorithm.

When the current conductor segment is followed by a node there are two possible results. If the node belongs to a subgraph, the conductor block may be developed into a subgraph block, assuming that the component has not already been placed in the layout. If the node is a circuit node, the conductor block in the base list is replaced by a node block which has the conductor as its two bound branches. The node is then developed as a normal base node.

Each node or conductor block of the base list is developed in turn. The working list then contains all the possible components, nodes and conductors that could be placed in the slot. These elements are also in the correct physical order within the list. It is possible that a component may appear more than once in the working list, as shown in Fig. 7.3(b). These multiple instances of components are removed in a later stage of the processing.

7.3.2 Orientation and Spacing of Components

All the components in a layout have the possibility of four different orientations. These correspond to each side of the component rectangle lying parallel to and facing the lower edge of

the slot. Before the spacing of components in a slot can be calculated, the component orientations must be determined. In addition it is necessary to know the number of conductors crossing under each component. This enables sufficient spacing to be allowed between adjacent components for these conductors to be routed to a higher level if necessary.

Every component in the working list is developed from a base node or conductor. The component pin to which the base node connects is termed the source pin. Each component is orientated so that its source pin is on the lower edge of the component, nearest to the base node. This reduces the length of conductors from the lower edge of the slot. As an example, the orientation of an integrated circuit component is illustrated by Fig. 7.4. The preferred

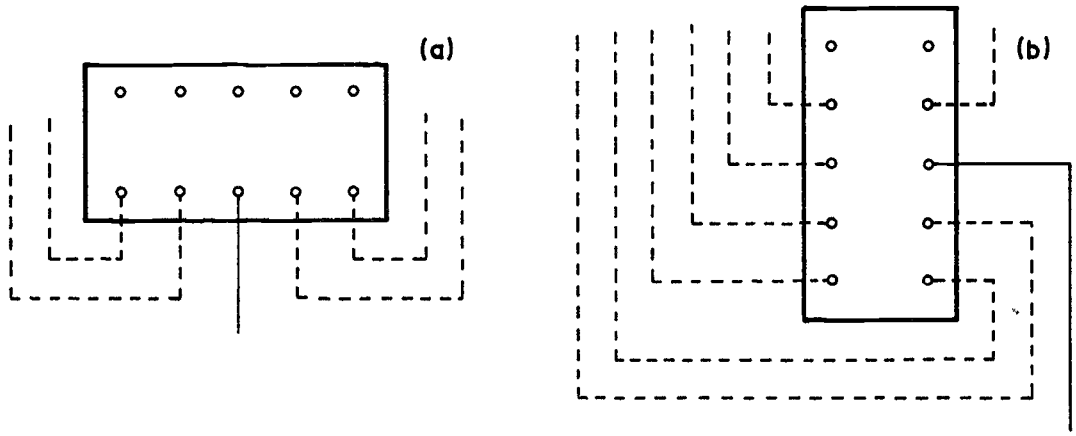


Fig. 7.4 Orientation of a subgraph component

orientation is shown in Fig. 7.4(a) whilst Fig. 7.4(b) shows the extra conductor routing required if the component source pin is not orientated towards the base node. If the source pin lies on a corner of the component there is a choice of two possible orientations. In such a case, the component is orientated towards an adjacent conductor which crosses under it, if one exists.

When the layout algorithm is operating automatically, each subgraph remains fixed in its orientation once this has been determined. This avoids the necessity of having to provide extra conductor routing such as that shown in Fig. 7.4(b). Each branch component is initially orientated with a shorter edge parallel with the bottom of the slot. This enables the maximum number of components to be placed in the slot. If there is space to spare in the slot, each branch component may later be re-orientated so that a longer edge is parallel with the bottom of the slot.

When calculating the spacing required for conductors to pass between adjacent components, it is assumed that conductors pass under components only at their crossing points. For example, components are spaced as shown in Fig. 7.5(a) as opposed to Fig. 7.5(b). This

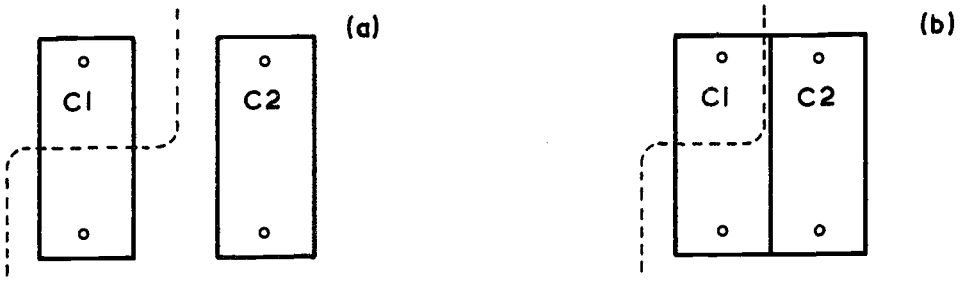


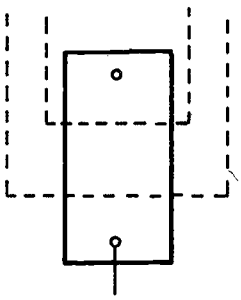
Fig. 7.5 Spacing of components

results in a slightly greater spacing than is necessary but avoids having to compute the positions of all the pins of the two adjacent components.

An algorithm has been developed to determine the orientation of a component and the spacing required for conductors which cross under, or are connected to it. The first operation of the algorithm is to identify the source pin of the component. A branch component

may then be orientated with its source pin lowermost in the slot. For a subgraph component it is necessary to know on which of the four sides the source pin lies. This is determined by examining four pointers which are stored in the master component block. These pointers indicate the four pins which are nearest to the corners of component. Given the source pin therefore, the corresponding side of the component may be determined and hence the appropriate orientation.

The spacing required for conductors around a branch component is calculated by counting the number of conductors which cross under it. This gives the left and right hand spacing required in the X direction. No spacing is necessary in the Y direction below the component. Neither is spacing required above as the top of the component will form the lower edge of a later slot. An example of conductor spacing is shown in Fig. 7.6.



LH. X spacing = 2 conductor widths
LH. Y " = 0 " "
RH. Y " = 0 " "
RH. X " = 2 " "

Fig. 7.6 Branch component conductor spacing

The spacing required around a subgraph component is computed in several stages. Firstly, the pin at the top left hand corner of the component is obtained. The number of nodes and pseudo branch crossings is then counted from this pin down to the pin at the bottom left hand corner. From the corner, the number of nodes and crossings is counted as far as the source pin. The second figure gives the left hand Y spacing required and the sum of the two

figures gives the left hand X spacing. Continuing the count to the bottom then the top right hand corner pins gives the corresponding values for the right hand side of the component. An example of conductor spacing is shown in Fig. 7.7.

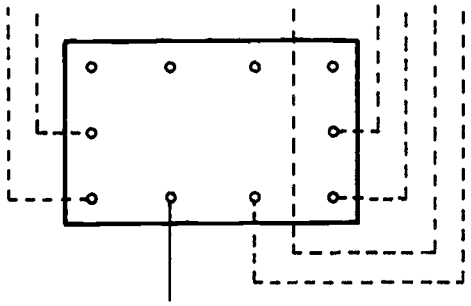


Fig. 7.7 Subgraph component conductor spacing

7.3.3 Counting of Slot Contents

Having developed the base list of a slot, the working list contains all possible components and conductors that could be placed in the slot. The next stage is the calculation of the total width of all these elements so that it may be compared with the actual width of the slot. In addition, some initial sorting of the working list is performed. This sorting is intended to remove multiple instances of components and unnecessary conductor paths.

The slot space occupied by a component is assumed to include space for conductors crossing under or connected to the component as well as the width of the component itself. In many cases, the conductors which are to cross under the component have already been developed from a lower level so that the working list contains their conductor blocks adjacent to the component block. These conductors are termed adjacent crossing conductors. The sum of block widths in

the working list would thus effectively include each of these conductor widths twice in the total. To avoid this, the destinations of conductors on either side of a component are checked before adding the component width to the total. Any adjacent crossing conductors are counted and each is given a special marker. The left and right hand spacings of the component are then reduced by the appropriate number of conductor widths.

The widths of all components, their left and right hand spacings, and all conductors are added together to give the total width of all the slot elements. At the same time a check is made for multiple instances of each component in the working list. When more than one instance of a component is found, the one with the greatest number of adjacent crossings is retained in order to minimise conductor lengths. The remaining instances are deleted from the working list and the total width of the slot contents is reduced accordingly.

When deleting a component from the working list it must be replaced by a node block. This preserves the connection from the base node to a further instance of the component. The bound branches of the replacement node are obtained by reference to the base node and any adjacent components connected to the same base node. If the working list already contains an instance of the node, adjacent to the component to be deleted, the bounds of the existing node block are merely updated.

It frequently occurs that a base node develops into several conductors. If these conductors do not cross under any components in the slot they are routed up towards a higher level slot. This would result in several parallel paths from one base node. To prevent

this, all conductors in the working list which have been developed from a base node and which have not been marked as crossing under adjacent components are replaced by their corresponding base node. Whenever two adjacent instances of a node then occur in the working list, the two node blocks are combined into one. The bound branches of the new node block are updated and the total width of slot contents is decremented by one conductor width.

The working list now contains one instance only of each component. All unnecessary parallel conductor paths have been removed and the total width of the potential slot contents is known.

7.3.4 Sorting of Slot Contents

The total width of the potential slot contents is compared with the actual width of the slot. There are three possible results, each with its corresponding course of action:

(a) The width of potential slot contents is greater than the slot width. Some components must therefore be removed from the working list.

(b) The slot is exactly filled by its contents. The algorithm may then proceed to the placement and routing stage.

(c) The slot width is greater than the potential contents. The spare space may be filled by reorientating some of the components.

A sorting algorithm has been developed to decrease the contents of the working list. The basic strategy is to keep the larger components in the list and to delete the smaller ones. This is based on the assumption that the smaller components may be more easily placed in later slots, especially if the later slots have less width than the current slot.

The first step is to take the value of the actual slot width and subtract from it the width of all nodes and conductors in the working list. This gives a figure for the maximum possible space available for components. In actual fact the available space is less than this because components are replaced by nodes when they are deleted. The insertion of these extra nodes into the working list gives rise to some difficulty in calculating the exact space available in the slot. When several components are developed from one base node there are many different combinations in which components, and nodes from deleted components, may occur. Fig. 7.8 shows just one sequence by which three components may be successively deleted from the working list.

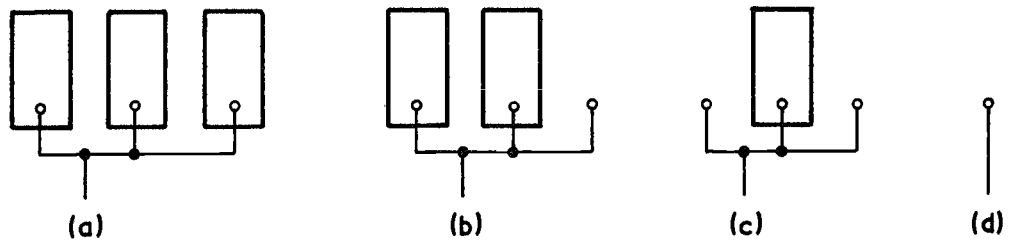


Fig. 7.8 Deletion of components from the working list

During the sorting procedure components may be marked to indicate that they are to be placed in the slot. The working list is searched for the largest component which has not yet been so marked. If any component is found that has greater width than the available space it is immediately deleted. Having found the largest component it is temporarily marked so as to keep it in the working list. The total width of the slot contents is then counted, replacing all unmarked components by the width of their base node.

If the total width is greater than the actual slot width, the component is deleted from the list. The total width of the slot contents has to be counted each time a component is placed due to the difficulties mentioned above. The search procedure is repeated until all the components have been either marked for placement in the slot or deleted from it.

A second sorting algorithm has been developed to increase the width of the slot contents. The strategy in this case is to reorientate branch components so that their longer sides are parallel with the bottom of the slot. This increases the space utilisation in the current slot and leaves a greater area of free space on the board for later slots. Preference is given first to components with adjacent crossing conductors. By reorientating a component towards its crossing conductors, the slot packing density can be increased and conductor lengths reduced. Second preference is given to components which have the greatest increase in space required when re-orientated.

The algorithm starts by calculating the extra width that each branch component would require if it were to be re-orientated. The calculated value is stored in the corresponding component block of the working list. The list is then searched to find the branch component with the greatest number of adjacent crossing conductors. A component found at any stage of the search whose increase in width due to re-orientation is greater than the spare space available in the slot is given an appropriate marker. It is then ignored in all further searches. Having completed the search, the resultant component is re-orientated towards its crossing conductors and the spare slot space is reduced accordingly. The component is then

marked so as to be ignored in further searches. The procedure is repeated until either the slot is completely filled or there are no further branches with adjacent crossing conductors.

The algorithm continues if there is still space to spare in the slot. The remaining unmarked branch components in the working list are examined to find the one which will give the greatest increase in width when re-orientated. The component thus found is re-orientated and the spare slot space is decreased accordingly. The search procedure is then repeated until either all the branch components have been re-orientated or the slot space is completely filled.

At this stage the working list is completely processed with reference to its contents. The components in the list are orientated for the most efficient use of the slot space and all the components and conductors in the list may be placed within the actual width of the slot.

7.4 Placement and Routing

The components and conductors to be placed in the slot are held in the working list in the correct physical order. They have resulted from the development and sorting procedures described in the previous section. The next stage of the layout algorithm involves the assignment of physical co-ordinates to the contents of the slot. Conductors may then be routed from the base nodes to the appropriate component pins and to the end of conductors held in the working list.

7.4.1 Component and Conductor Placement

The conductor blocks in the working list represent conductors which are to be routed from a base node at the bottom of the slot, through the slot and up to a later slot at a higher level. Each conductor end is assigned a physical co-ordinate so that it may be projected upwards to a higher level without meeting an obstruction. The conductor ends are therefore assigned co-ordinates in exactly the same manner as components.

The X co-ordinates are assigned by working across the slot from left to right. The initial X co-ordinate is set to the left hand edge of the slot. The first component or conductor is then positioned at this co-ordinate. In the case of a component, due allowance is made for the space required by crossing conductors. The X co-ordinate is then increased by the total width of the element just placed. This enables the procedure to be repeated with the remaining components and conductors in the slot.

When assigning the Y co-ordinates of the slot contents, several points must be taken into consideration. The first is illustrated by the example in Fig. 7.9. In routing a conductor path from a base node to its appropriate conductor end, it may have to pass over several other base nodes. The conductor end must therefore be given sufficient Y clearance from the bottom of the slot to enable all the conductor paths to be routed without intersections. Similarly, components require clearance from the bottom of the slot in order to prevent unwanted conductor crossings.

It may be observed from Fig. 7.9 that nodes B, C, D and E have to be routed around node F. This node is therefore the basic obstruction to the routing of the other nodes and it causes a

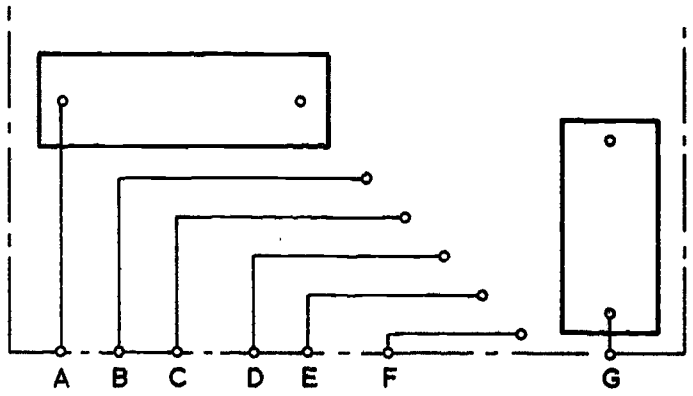


Fig. 7.9 Placement of components and conductor ends

"wave front" of conductor corners to the left of itself. Each corner point is one conductor width to the left and above the previous corner. This fact is used in calculating the Y displacement of components and conductors.

To calculate the required Y displacement of a component or conductor, the right hand X co-ordinate of the element together with its base node are first obtained. The next base node to the right is then examined and its co-ordinates obtained. The position of the "wave front" caused by this node may thus be calculated. The procedure is repeated with successive base nodes to the right until either the co-ordinates of the "wave front" lie to the right of the current component or the end of the base list is reached. The number of base nodes examined indicates the required number of conductor-width displacements of the component in the Y direction. The whole procedure is then repeated on the left hand side of the current component or conductor. The larger of the two figures gives the required Y displacement.

Before placing a component in the slot, a further Y displacement may be necessary due to crossing conductors along the lower edge of the component. If the component has both a left and right hand Y displacement for crossing conductors, the larger of the two is taken. This is then added to the Y displacement described above to give the total displacement of the component. An example of such component placement is shown in Fig. 7.10. When the total

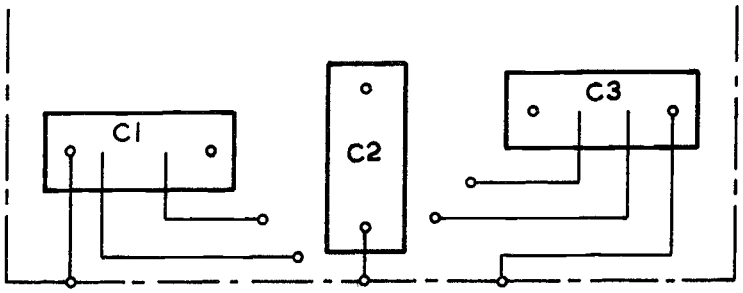


Fig. 7.10 Placement of components

displacement of each component in the slot has been calculated, it is assigned a Y co-ordinate and added to a list of placed components.

During the placement of elements in a slot, it frequently occurs that the last few elements are conductors followed by spare space at the right hand side of the slot. Conductor ends that are placed to the right of their respective base nodes have to be routed around components as shown in Fig. 7.11 (a). Conductor ends that are placed to the left of their respective base nodes have no such obstacles to avoid. In addition, if these conductors have to be routed to the right in a later slot they will follow an un-necessarily long path as shown in Fig. 7.11 (b).

When the components and conductor ends have been assigned co-ordinates, the working list is scanned from the right hand side.

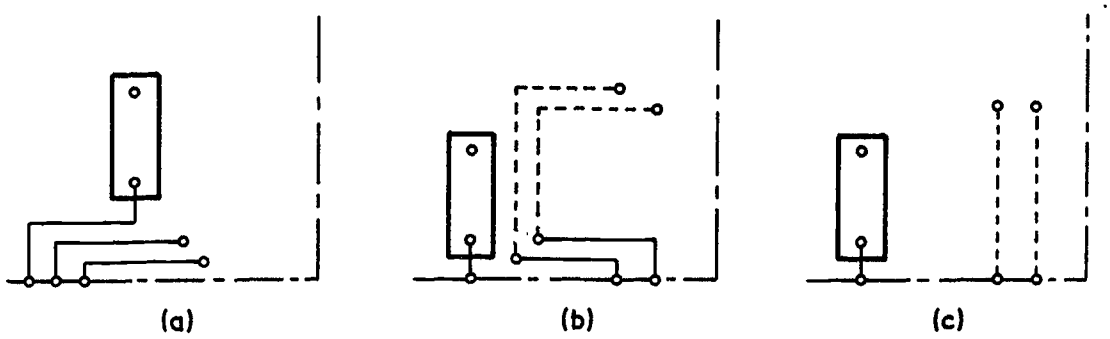


Fig. 7.11 Placement of conductors at RH. side of slot

If a conductor end is found which lies to the left of its base node, it is repositioned to the same co-ordinates as its base node. This results in shorter conductor paths as shown in Fig. 7.11 (c). The scanning of the working list is continued until a component block is encountered, or a conductor which is routed to the right from its base node.

7.4.2 Placement of Crossing Conductors

At the stage now reached in the processing, the components have been placed in the slot. The conductors which cross under or are connected to these components may therefore be placed in the layout. The routing procedure is performed in two stages. Firstly, a list of all the nodes and crossing conductors around a given component is constructed by referring to the topological model. The actual crossing conductors are then routed. In the second stage, the nodes and conductors are routed out around the component and their list is connected into the working list.

To process a subgraph component, the first pseudo branch of the component is obtained. By referring to the topological model, the number of crossings of this branch, if any, may be determined.

By further reference to the component position and its master block in the component library, the co-ordinates of the two end points of the pseudo branch may be calculated. The co-ordinates of the required number of crossing points, equally spaced along the branch, may thus be calculated. At the same time, a list of blocks is constructed, containing a node block for the first node of the pseudo branch and a conductor block for each of the crossing conductors.

The procedure is repeated for each pseudo branch in turn, adding node or conductor blocks to the end of the list as they are encountered. The two ends of the list are then joined to form a ring, for reasons explained below. The ring thus contains all the nodes and crossing points, with co-ordinates, in the same order as would be obtained by traversing the perimeter of the component rectangle. An example of component crossing points is shown in Fig. 7.12. The pins of the component are labelled A to F and the

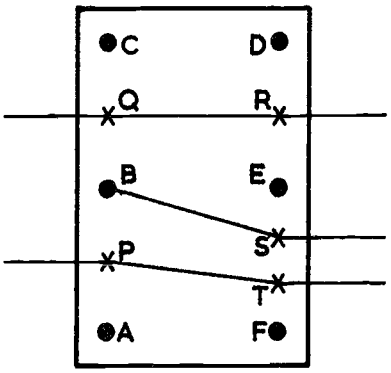


Fig. 7.12 Subgraph component with crossing conductors

crossing points are labelled P to T. The order of blocks in the ring would thus be : A P B Q C D R E S T F.

The next step is to route the crossing paths under the component, for example paths Q to R, B to S and P to T in Fig. 7.12. Each block in the ring is examined in turn. When a conductor block

is encountered the topological model is checked to find the node or crossing conductor to which it is connected. The ring is then searched to find the corresponding node or conductor block. The co-ordinates of the two points may thus be obtained and a conductor path routed between them. The procedure is repeated for the remaining conductor blocks in the ring so that all the crossing paths are routed under the component.

Branch components are processed in a similar manner. A ring of nodes and crossing conductors is constructed as before. In this case the conductor blocks may be matched in pairs, corresponding to a crossing conductor appearing on two sides of the component. The co-ordinates of the two blocks in each pair are identical so no conductor routing is required under the component.

The conductor crossing procedure is repeated for every component in the working list so that each has a ring of node and conductor blocks associated with itself. The next stage of processing involves routing the node and conductor paths around the component as part of the layout procedure. Also the ring of blocks associated with each component has to be connected into the working list.

The conductor routing algorithm described later is based partly on the assumption that conductors may always be projected up to a higher Y level without encountering any obstruction. When a component is placed it is necessary to route its connected nodes and crossing conductors so that this assumption is true. An example of routing is shown in Fig. 7.13. Nodes and conductors on the sides of the component are routed outwards to the left or right. Those

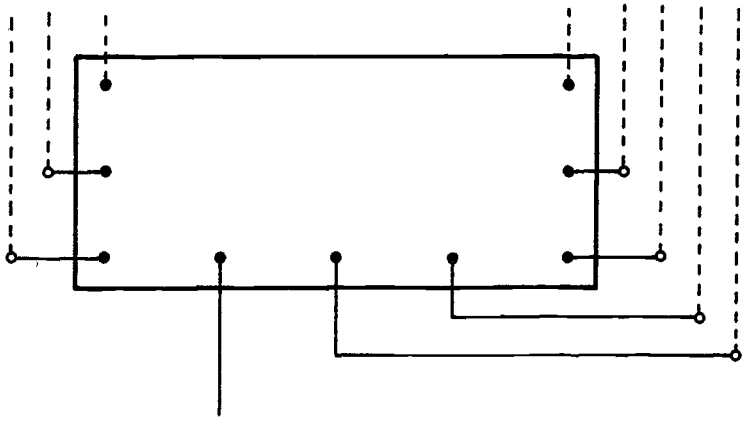


Fig. 7.13 Routing of component nodes

along the bottom edge of the component are routed downwards then outwards as shown in the diagram.

A further complication occurs when the component has adjacent crossing conductors or nodes as illustrated in Fig. 7.14.

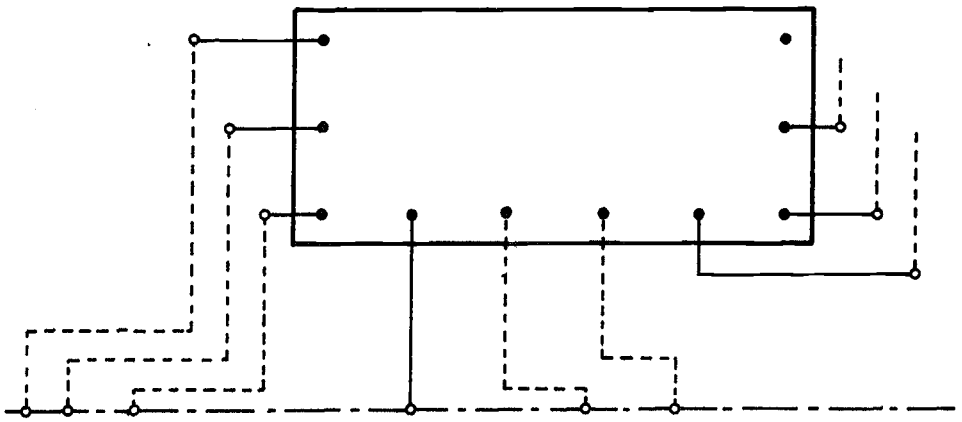


Fig. 7.14 Routing of adjacent component nodes

Nodes on the lower edge of the component do not need to be projected outwards. Conductor paths are routed up towards them from the base level at a later stage of the procedure. Nodes on the side edges of the component however, have to be projected outwards to different X co-ordinates as shown. This prevents

intersections when the conductor paths are routed up from the base level.

The procedure for routing the nodes and conductors outwards from a component starts by searching the ring of blocks for the one corresponding to the component source pin. The blocks in the ring to the left of the source pin are then examined in turn. Any adjacent connected nodes on the lower edge of the component are passed over. The remaining nodes on the lower edge, if any, are routed downwards by the required amount so that they may later be routed sideways without intersection. The reason for forming the blocks into a ring is that the block corresponding to the source pin may occur at any point in the list of nodes and crossing conductors. The routing procedure has to examine blocks both to the left and to the right of the source pin. It is thus more easily programmed if the blocks are connected into a ring instead of a straight list.

The nodes which have just been routed downwards, together with those on the left hand side of the component, are examined in turn. Each node is routed out to the left of the component so that its X co-ordinate differs by one conductor width from that of the previous node. The difference is negative if the node is to be connected to one in the base list and positive if the node is to be routed up to a later slot level. As each node is routed the co-ordinate of its end point is updated. The whole procedure is then repeated for nodes and conductors to the right of the source pin.

At this point, the blocks representing adjacent conductors and nodes of a component have been duplicated by the various layout algorithms. One instance of each block appears in the working list,

developed from a block in the base list. The other instance appears in the ring of blocks associated with the component. To remove one instance of each block from the data structure, the co-ordinates of the blocks in the working list are first re-assigned to the co-ordinates of the corresponding blocks in the component ring. The duplicate blocks are then deleted from the component ring.

For each component in the slot, the ring of blocks is split at the source pin block so that a straight list is formed. This list is then inserted into the working list adjacent to the component block. The working list thus contains all the nodes and crossing conductors around each component in addition to the elements which it previously contained. Furthermore, the order of these nodes and conductors still corresponds to the order of those encountered in scanning across the slot from left to right.

7.4.3 Processing of Base List Elements

The working list of the slot includes at this stage a number of node and conductor blocks which have been developed from the base list. It also includes a source node block for each component in the slot. The conductor routing procedure routes paths from each element in the base list to one or more of these elements in the working list. Before the routing can proceed however, the appropriate blocks to which each base element is to be connected to must be identified. In addition, pairs of elements in the base list may correspond to two parts of the same node or conductor. These parts must be identified so that they may be connected together.

Node and conductor blocks in the working list which are connected to a base node are termed the target blocks of the base

node. The first node of the base list is taken and a note is made of the next node in the list. The working list is then scanned from left to right to find all the elements connected to the first base node. As each target block is found it is added to a list of targets associated with the base node. The scanning of the working list is continued until an element is found which is connected to the next base node. The procedure is then repeated for each base node in turn until every one has an associated list of target blocks. Nodes and conductors in the working list which have been inserted from a component ring do not have a base node and thus are not routed at this stage.

Having found the target blocks of each base node, the base list is checked for pairs of elements belonging to the same node or conductor. In the case of a node there may be more than two blocks belonging to the same node. A typical arrangement of connected base elements is shown in Fig. 7.15. It may be noted from the diagram

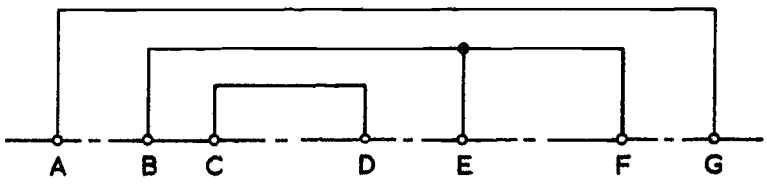


Fig. 7.15 Connection of base nodes and conductors

that successive sets of connected base elements are "nested" around the previous sets. To check the base list, each block is compared with the next block to the right until two are found belonging to the same node or conductor. As an example, the blocks C and D in Fig. 7.15 would be found first. A marker is set in the two base blocks and they are further processed in the manner to be described

below.

The process of comparing each base block with the next is repeated. If however, a marked block is encountered when looking for the next block, it is passed over and the following block is examined. Referring to Fig. 7.15 again as an example, it is assumed that blocks C and D have already been marked. When the next block following B is searched for, blocks C and D will be passed over so that blocks B and E are compared and marked as part of the same node. During the following search, blocks C, D and E will be passed over when finding the next block after B. Thus block F will be identified as another part of the same node. The comparison procedure is continually repeated, identifying another pair of connected blocks at each pass through the base list. It is completed when a complete search is made through the base list without finding another connected pair of blocks.

The processing of a connected pair of base blocks involves the checking and modification of several elements of data. Consider first the connection of two parts of a conductor, such as blocks C and D shown in Fig. 7.15. Each conductor block in the base list has a corresponding block in the working list to which a path will be routed by the conductor routing algorithm. The two conductor blocks in the working list are therefore modified so that their corresponding base blocks will be connected. The co-ordinates of both the blocks in the working list are re-assigned to the co-ordinates of the left hand base block. The conductor routing algorithms will thus construct a path from the right to the left hand base block.

The connection of two parts of a node is more complex as not all the parts of the node may yet exist in the layout. The

co-ordinates of the two working list blocks are updated in the same manner as the conductor blocks described above. In addition, the bound branches of the two working list node blocks are checked. The left hand bound branch of the left hand part of the node is examined first. The topological model is referenced to find the next branch on the node in a clockwise direction. If this is the same as the right hand bound branch of the right hand working block the node is complete. If the node is not yet complete the bound branches of the remaining part are stored in the left hand working block.

An example of the connection of two parts of a base node is shown in Fig. 7.16. Part (a) shows the topological representation

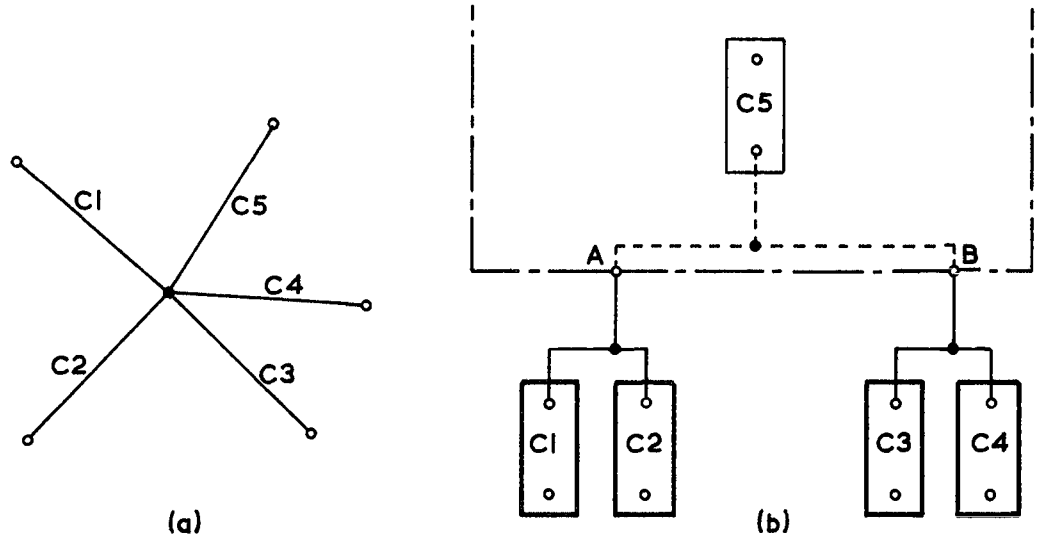


Fig. 7.16 Connection of two parts of a node

of the node and its attached components whilst part (b) shows the partial layout of the node and components. It is assumed that the base blocks A and B have been recognised as two parts of the same node. The left hand bound branch of block A is then found to be component C1. Referring to the topological model, the next branch

in a clockwise direction on the node is component C5. The component does not correspond to the right hand bound branch of block B, which is component C4. This indicates that a further connection has to be made to the node, in this case component C5. When the connections to a node are thus not complete, the base blocks are specially marked. This prevents any further connections from being nested around these base blocks.

7.4.4 Routing of Conductors

The blocks in the base list have now been prepared for the actual routing of conductors. Each base block has a list of the working blocks to which it is to be connected and each working block has been assigned its appropriate co-ordinates. The basic principle of the conductor routing algorithm is that each conductor is routed towards its target X co-ordinate and then up to its target Y co-ordinate. The conductors are constructed by operating in strips parallel to the bottom edge of the slot and one conductor width wide. If a conductor meets an obstacle during routing, such as another conductor, it is projected up to the next strip level and the routing is attempted again at the next level.

An example of the method of conductor routing is shown in Fig. 7.17. It can be seen that the resultant conductor paths are orthogonally routed, i.e. all parts of each path are parallel with either axis of the rectangular board perimeter. The paths so produced are not generally the shortest possible between a base block and its targets. This method of routing however, has two major advantages in the construction of conductor paths. The first is that the tedious calculation of clearances between adjacent

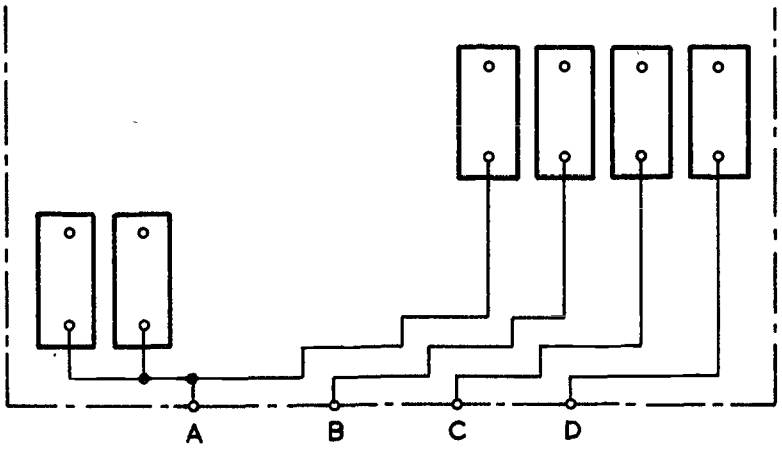


Fig. 7.17 Routing of conductors

conductors at different angles is avoided. The second advantage is that the components are already placed to allow sufficient clearance for orthogonally routed conductors. No checking of component positions is therefore necessary during routing.

During the routing of conductors, it is necessary to know the current end point of each conductor path so that intersections may be avoided. A base node may be routed both to the left and to the right from its initial position as illustrated by node A in Fig. 7.17. To store the current conductor end points therefore, each base node block has two base limit elements. These store the X co-ordinates of the end points on either side of the base node during routing. Initially the two base limits are set to the X co-ordinate of the base node itself.

When a conductor path has been successfully routed to a target block in the working list, there are two possible ways of dealing with the block. If it represents a specific point such as the source pin of a component, the routing to that point is complete. The block is therefore deleted from the working list. The other

possibility is that the target block represents the end point of a node or conductor which is later to be routed up to a higher slot. In this case the block is retained in the working list so that it may be included in the base list of the later slot.

The conductor routing algorithm starts by routing up to the lower edge of the slot any base nodes which are below this level. The first level of routing is then carried out, taking successive base nodes across the slot from left to right. The node to be routed next is selected from the base list. Its list of targets is searched to find the one nearest to the base node and on its left hand side. The base block to the left of the current block, if any, is then examined to find its right hand base limit. This is compared with the chosen target X co-ordinate to check for possible obstruction of the conductor path.

If the path to the target block is not obstructed, a conductor is routed first horizontally then vertically from the base node to the target. This is illustrated by the components to the left of node A in Fig. 7.17. The target block is removed from the list of base node targets and is also deleted from the working list if necessary. The left hand base limit of the base node is then updated to the X co-ordinate of the target. A different procedure is employed if the path to the target block is obstructed. A horizontal conductor is routed from the base node to within one conductor width of the obstruction. It is then routed up one conductor width to the next strip level and the base limit is updated to the X co-ordinate of the current conductor end. The routing of the path is continued later at the next strip level.

Any further target blocks to the left of the base node are routed in turn, assuming there are no obstructions. Each new conductor path starts at the current co-ordinate of the left hand base limit as shown in Fig. 7.18. The routing of conductor paths to

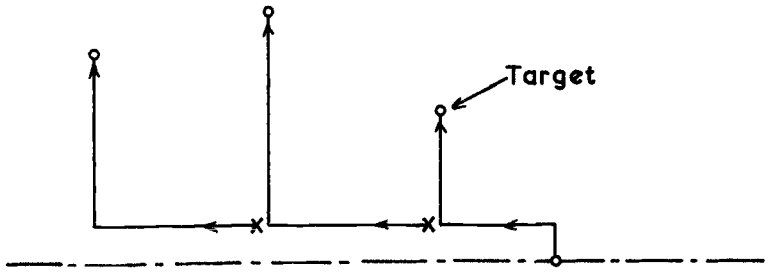


Fig. 7.18 Conductor routing to targets

successive targets continues until either an obstruction is encountered or there are no further targets to the left of the base node. The routing procedure is then performed in a similar manner for targets on the right hand side of the base node. The same routing process is then carried out for each base node in turn across the slot.

At the end of one pass across the slot, some of the base nodes may have an empty target list. All the targets have been successfully connected to each of these nodes so they are deleted from the base list. The remaining base nodes have all been obstructed at some stage of their conductor path routing. Base nodes which have been routed in one direction only, such as nodes B, C and D in Fig. 7.17, have both their base limits set to the X co-ordinate of the current conductor end. The conductor routing level is then incremented by one conductor width and the routing procedure is repeated with each of the remaining base nodes in turn. The whole procedure is repeated at successive routing levels until no blocks remain in the base list.

The processing of one slot is completed at this stage. The components and their crossing conductors have been placed in position and all conductors within the slot have been routed. The slot base list is empty and the working list contains the component blocks and any remaining nodes or conductors which are to be routed up to a later slot. The processing of the remainder of the working list is described in the next section.

7.5 Overall Layout Algorithm

This section describes the algorithm for the overall control of the layout process. It deals basically with the organisation and selection of successive slots, each of which is processed in the manner previously described.

7.5.1 Selection of Slots

The width and co-ordinates of successive slots are determined by the components which have already been placed on the board. This principle is described earlier in section 7.2. To facilitate the computation of these slot dimensions and co-ordinates, a list of components placed on the board at the current working level is constructed. The order of components in the list corresponds to their order across the board. The list also contains the X co-ordinates of the two sides and Y co-ordinate of the top edge of each component. An example of component positions is shown in Fig. 7.19. Components C1 to C7 are placed components at the current working level from which the position of the current slot has been calculated. Components C8 to C10 are part of the current slot and will be added to the placed component list at a later stage.

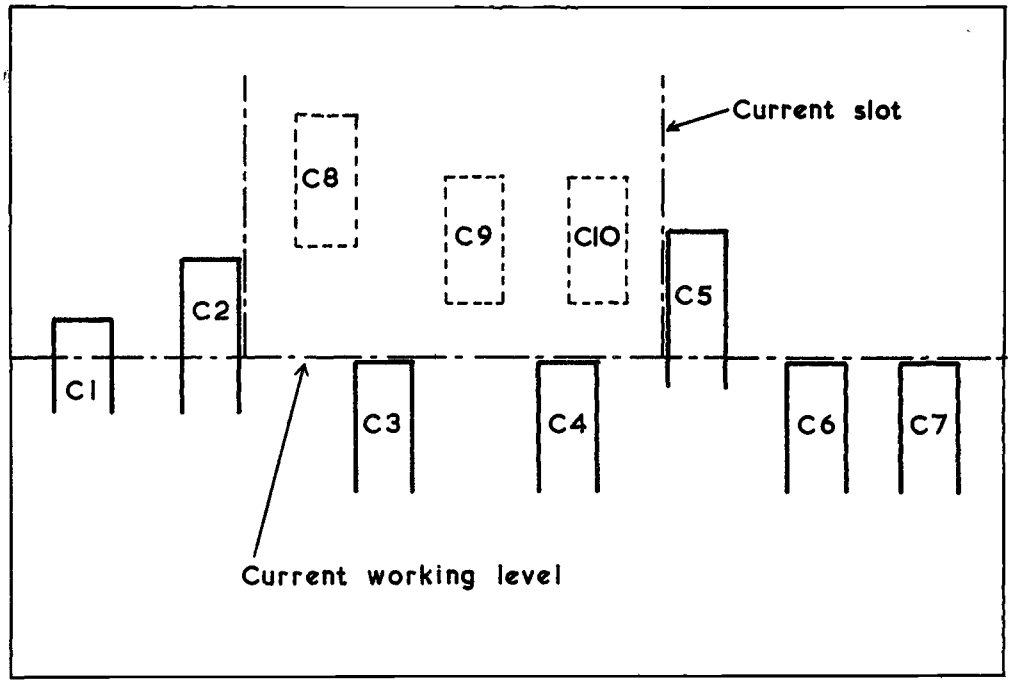


Fig. 7.19 Placed component list and selection of slots

The co-ordinates of the next slot to be processed are found by examining the placed component list. The list is first searched to find the component with the lowest upper edge. This determines the bottom edge, or working level, of the slot. The left and right hand X co-ordinates of the slot are then coincident with the two component sides or board edges which project above the working level on either side of the lowest component. If there are several possible slots at the same level, the leftmost slot is chosen first. The choosing of slot boundaries is illustrated by Fig. 7.19. Components C3, C4, C6 and C7 are all at the current working level. Component C3 is taken first, being the leftmost component. This then gives the positions of the slot sides as the sides of components C2 and C5.

When all the components have been positioned in the current slot, the placed component list is updated. The list is first

searched to find the two components which lie on either side of the slot. The intervening components thus lie below the current working level and so are deleted from the list. The newly placed components are then inserted into the same part of the list. In the example of Fig. 7.19, components C3 and C4 lie below the current slot. When they are deleted from the list, components C8, C9 and C10 are inserted between components C2 and C5.

7.5.2 Description of Flow Diagram

The flow diagram for the overall layout algorithm is shown in Fig. 7.20. The algorithm starts with several initialisation procedures. These include the initialisation of the free storage system described in Chapter 9.2 and the setting up of dummy end blocks for the base, working and placed component lists. The board dimension data is then read in. It consists of the board length and width together with the X co-ordinate of each edge connector pin across the lower edge of the board. The initial base list is then constructed by referring to the topological model of the layout. The outside edge of the graph gives the list of edge connector nodes in the correct order. The bound branches of these base nodes are given by the two pseudo branches connected to each node.

The boundary of the initial slot is made coincident with the sides and lower edge of the board. The base list is then developed in the manner described in section 7.3.1 to form the working list. The contents of the working list are then processed and sorted as previously described. The components, if any, are positioned in the slot and their crossing conductors are routed under the components. The placed component list is then updated and the corresponding component blocks deleted from the working list.

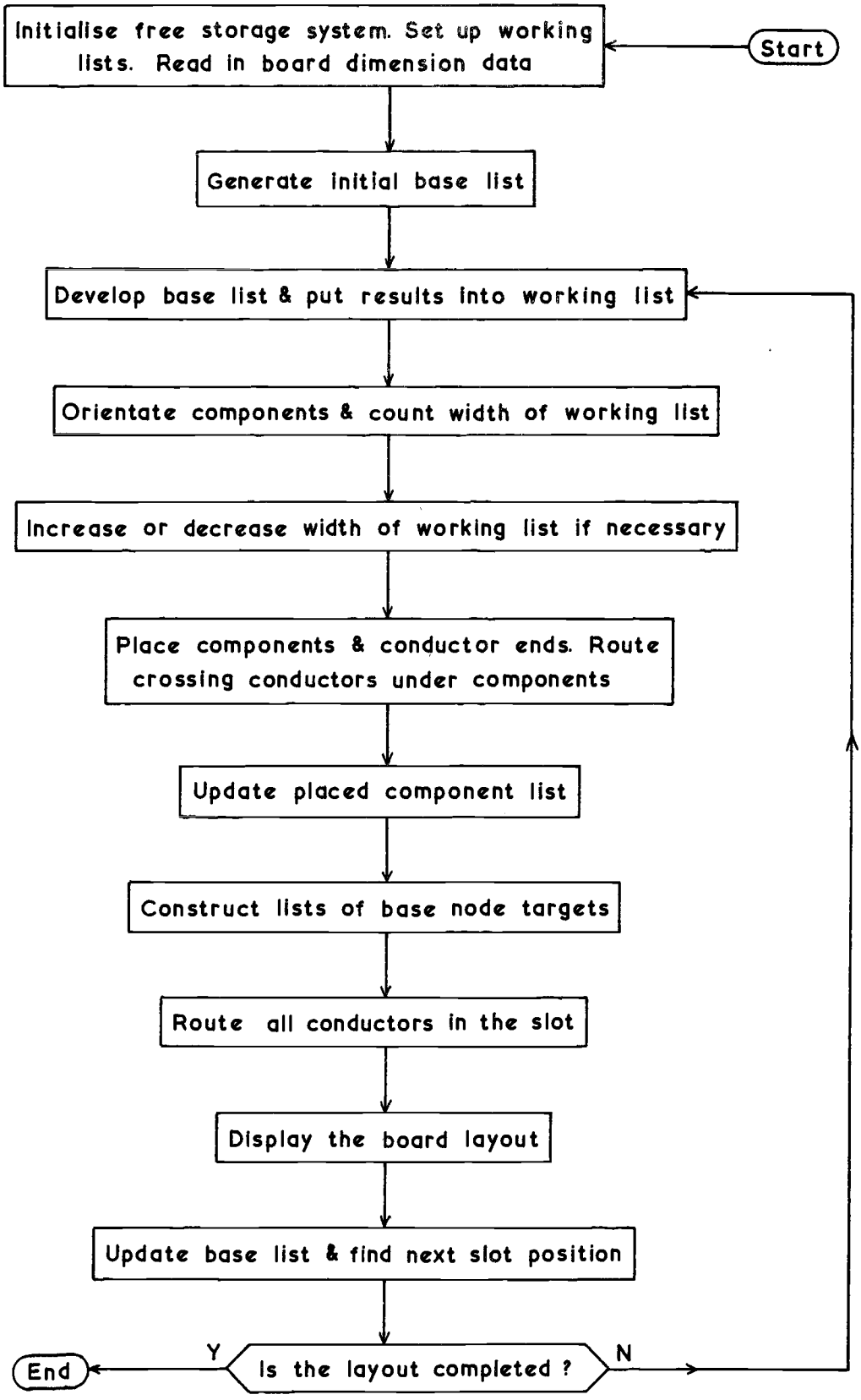


Fig. 7.20 Flow diagram of layout algorithm

Before proceeding, a check is made for a set of conditions which may occur in a similar way to those illustrated in Fig. 7.21.

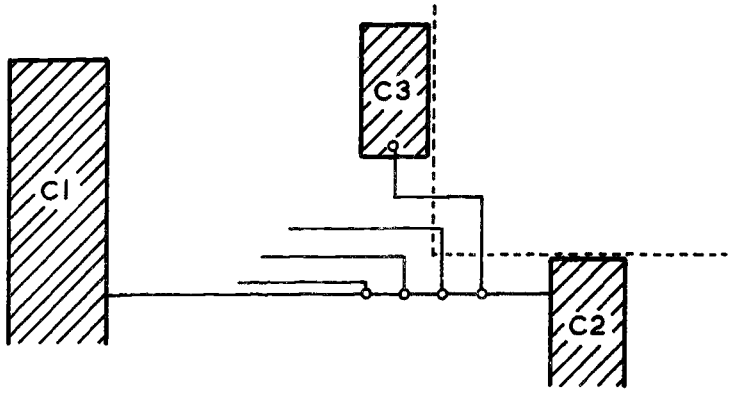


Fig. 7.21 Conflict of conductors in slots

The current slot is bounded by components C1 and C2. Component C3 is placed in the slot, displaced upwards by a number of conductors as shown. A later slot will then be bounded by components C2 and C3 as shown by the dotted line in the diagram. The left hand corner of the later slot will contain some conductors from the current slot so that conflict of component and conductor placement may occur. To prevent this happening, a dummy component is inserted into the placed component list to coincide with the offending conductor at the right hand side of the current slot. This action is only necessary if the highest Y co-ordinate of the end conductor is greater than the working level of the later slot.

The layout algorithm then proceeds to the insertion of conductor paths. The list of targets for each base node is first constructed then all the conductor paths of the slot are routed. Having completed the placement of components and conductors in the current slot, a display of the current board layout is generated. The display is used for the interaction procedures to be described later and its method of generation is described in Chapter 8.2.

At this stage a check is made for a set of conditions which may occur such as those illustrated in Fig. 7.22. The

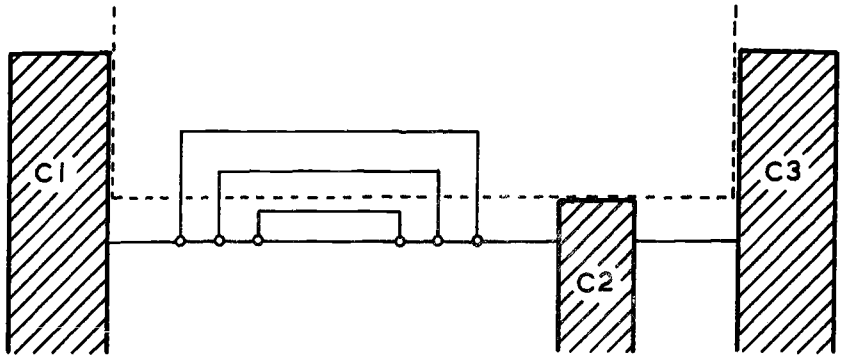


Fig. 7.22 Conflict of conductors in slots

current slot is bounded by components C1 and C2 and several of the base nodes are connected together as shown. A later slot will be bounded by components C1 and C3 with its working level at the top edge of component C2. The routed conductors of the current slot will thus lie within the boundary of the later slot. As slots are assumed to be initially empty, some conflict of conductor routing may occur. The solution to this problem is to update the upper level of all component blocks between and including the current slot limits. The upper levels in the placed component list are set equal to or greater than the highest level of conductor routing so that the later slot will lie above these conductors.

At this stage of the layout algorithm the working list of the current slot contains only the blocks of nodes and conductors which are to be routed up to a later slot. The base list, although not explained previously, contains all such nodes and conductors across the board at the current working level. Only a section of the list is used at any one time to form the base list of a slot.

The working list of the current slot is therefore inserted into the overall base list so that the nodes may be developed in a later slot. As the nodes are ordered from left to right across the board, the correct point of insertion of the working list may readily be determined.

The layout algorithm continues by examining the placed component list to find the position of the next slot. Having found its co-ordinates, the base list is searched to find the set of base nodes which lie between the sides of the slot. This set then forms the base list of the next slot so that the whole procedure of processing a slot may be repeated. The layout algorithm is completed when all components have been placed and all node and conductor interconnections completed. This state is detected when the base list of all nodes across the board is empty. The layout is then complete and is ready for output by the method to be described in Chapter 9.

Chapter 8 Interaction with Board Layout

Interaction is defined as the close communication between a computer program and the user, whilst the program is running. In terms of the board layout problem this means that the user can observe and alter the course of the program during the computation of a layout. Interaction thus enables the layout algorithm to be supplemented by the skill of the user and should result in layouts which are an improvement upon those produced by purely automatic methods.

The man-machine communication devices used are a graphical display for computer output, and a light pen and Teletype keyboard for input. Interaction with the layout program is feasible only if a graphical display is available to present the necessarily large quantities of visual data rapidly. Other forms of output either give insufficient detail, as in the case of a Teletype, or take an excessive time to produce useable data, as in the case of a mechanical plotter.

8.1 Objectives of Interaction

There are two aspects of the layout program in which interaction may be most usefully employed. They are situations in which the exact definition, and hence programming, of the problem is very difficult. The user, however, has the ability to examine the overall state of the layout and to intuitively find a solution to the particular part of the layout problem. He may then modify the layout accordingly by the use of interaction.

The first use for interaction is in satisfying special requirements for particular board layouts. Some boards may require

certain components to be specially positioned. For example, the adjustment screw of a potentiometer or variable capacitor should be accessible from the front edge of the board. Other boards may require certain critical components to be closely grouped together so that they may be attached to a heat sink and maintained at equal temperatures. Other boards again may require the input and output conductors of a high gain or high frequency amplifier to be kept as far apart as possible so as to reduce the effects of stray capacitance.

Conditions such as those just described are difficult to incorporate into a general purpose layout program. The obvious approach is to use an automatic layout algorithm to do most of the work in producing a board layout. The user then interacts with the algorithm in the areas where special conditions have to be satisfied.

The second use for interaction is in the improvement of an automatically-produced layout. The layout algorithm optimises the placement and routing of a succession of slots, or subsets, of the layout. The optimum placement for each slot however, may not be the optimum for the whole board. Interaction enables the user to assess and modify the overall appearance of the layout. By re-positioning a number of components it may be possible to improve the component packing density and reduce the total conductor length.

8.2 Generation of Display

The display of the current state of a board layout is an essential stage in the process of interaction so it is generated after each slot has been processed. A partially completed layout is shown in Fig. 8.1. The board outline is shown as a rectangle

P O D R U M F

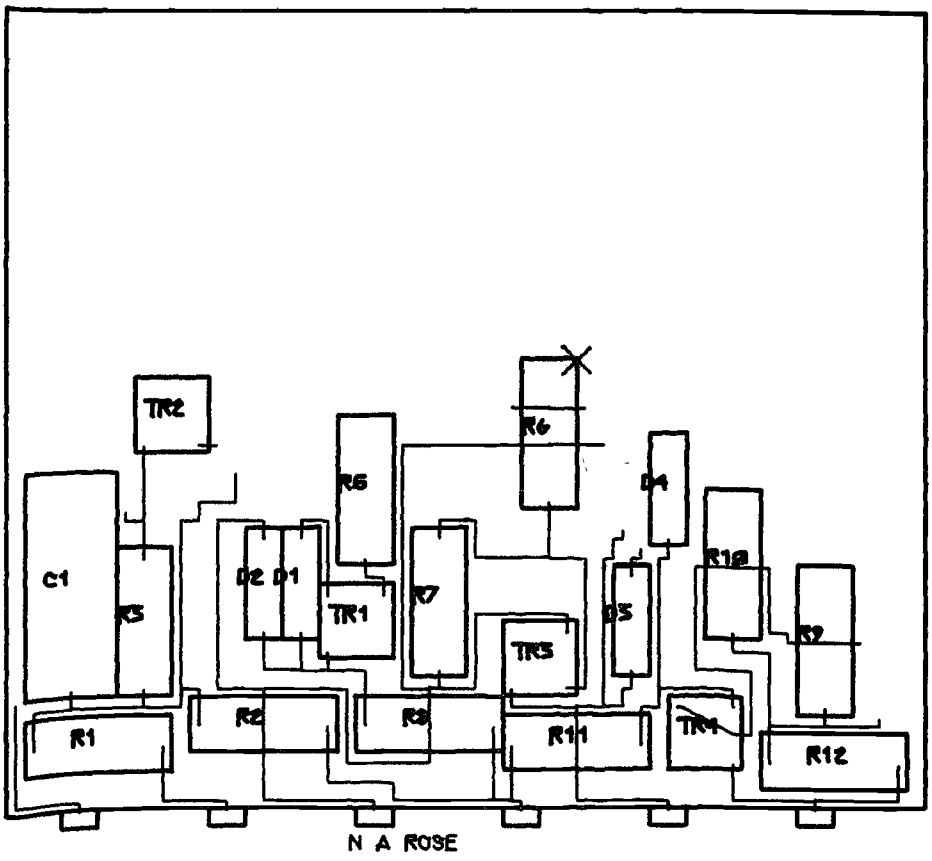


Fig. 8.1 Display of board layout

with the edge connector pins across the bottom edge. Each component is shown as a bounding rectangle labelled with the component name. Conductor paths are shown by lines representing the centre line of each path. On the display, components and conductors are drawn at different intensities so that they may be readily distinguished. Across the top of the display screen are a number of characters, or light buttons, that may be used to control the modes of interaction.

The display software, described in Appendix B, enables the display file to be divided up into a number of segments. Each segment may then be uniquely identified in the graphical display by pointing at it with the light pen. In addition, every segment may be assigned an integer number by the user, termed the user name. Each component and light button to be displayed is therefore generated as a separate segment so that it may be uniquely identified. In the case of a component, the user name is then used to provide a pointer back to the appropriate component block in the data structure.

The generation of display file is commenced by positioning the seven characters for the light buttons across the top of the screen. The user names for these light buttons are set to the integers one to seven so that they may later be identified and processed when seen by the light pen. The remainder of the display file to be generated has all of its dimensions multiplied by a display scale factor. This factor is read in as part of the board data and is used to ensure that the layout fills the display screen.

The next part of the display to be generated is the rectangle representing the board perimeter. The pins of the edge connector are then plotted in representational form across the

bottom edge of the board as shown in Fig. 8.1. In practice there is usually a standard mask which surrounds the actual board layout. This mask defines the board outline, the pins of the edge connector and any further information necessary.

Each component to be displayed is generated as a display subroutine so as to conserve display file space. Furthermore, the component will have one of four possible orientations. Every master component block in the component library therefore has four elements allocated for display. The elements contain pointers to the display subroutines for each of the four orientations if they have been generated; otherwise they contain a zero pointer.

To display a component, the beam position is set to the appropriate co-ordinate. The orientation of the component is obtained and the corresponding element of the master component block is checked. If that particular orientation has not yet been plotted, the required display subroutine is generated and its address stored in the master component block. The component is then plotted as a separate display segment together with its component name. The component name may consist of up to four characters, evenly spaced about the centre point of the component rectangle. This explains why names of less than four characters appear to be offset to the left. The user name of the display segment is then set as a pointer back to the component block in the topological model.

When all the component subroutines have been generated, the conductor paths are plotted. The conductors are held in a list and each one is represented by a list of change points, described in more

detail in Chapter 9.1. As there is no interaction with conductors they are all generated in one display segment. The display of a conductor is generated as a co-ordinate point at its start followed by a string of vectors describing the conductor path.

When the display file has been completely generated it is transmitted over the link to the PDP-7 computer. The display file is then shown on the graphical display so that the user may examine it and operate upon it with the light pen.

8.3 Interaction Facilities Provided

The light buttons on the display provide the user with a number of modes of interaction, which are described below. The modes of interaction are concerned with the movement of components only as these control the overall form of the layout. The automatic part of the algorithm then deals with the correct clearances and routing of conductors. A state diagram of the interaction subroutine is shown in Fig. 8.2. It illustrates the ways in which the user may change from one state, or mode, to another. The letters by each state indicate the light buttons to be activated in order to change to further states of the program.

8.3.1 DELETE Mode

The DELETE mode enables the user to delete a component from the slot in which it is placed. The component is removed from the slot and replaced by its source node. The source node will then be projected up to the level of later slots until there is a slot with sufficient space to accommodate the component. The effect of deletion therefore is to move components up to a higher level in the layout. The slot from which the component is deleted will have

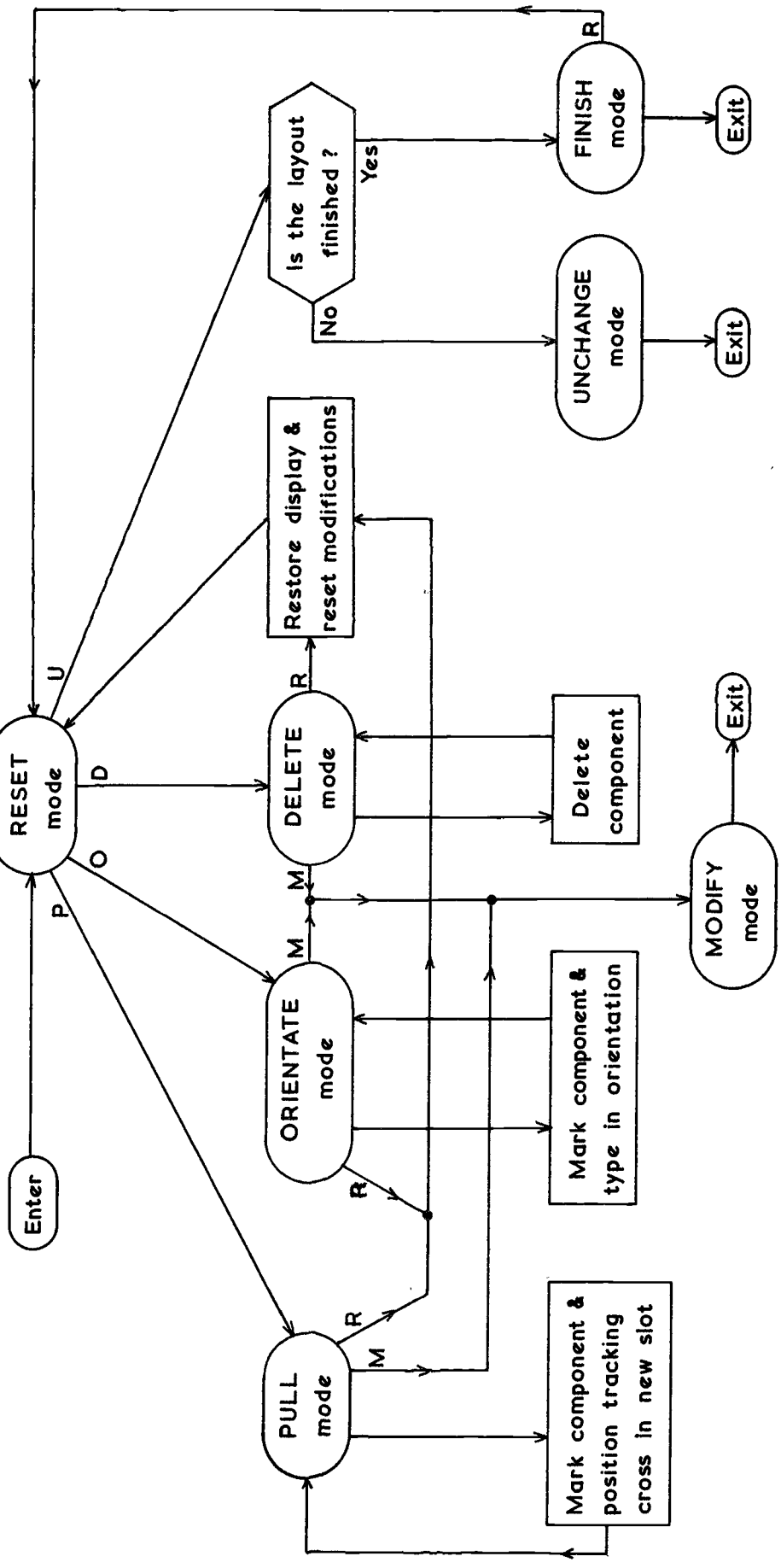


Fig. 8.2 State diagram of interaction subroutine

spare space corresponding to the component width. The layout algorithm will automatically attempt to fill this space by inserting further components or re-orientating the existing components in the slot.

To operate the DELETE mode, the user points the light pen at the light button "D". The light button character is then displayed at twice the scale to indicate which mode the program is in. The user then points at the component to be deleted, which immediately disappears from the display. The modification may then be implemented by entering the MODIFY mode described below.

8.3.2 ORIENTATE Mode

The ORIENTATE mode enables the user to alter the orientation of components in the layout. There are some restrictions on the number of orientations that each component may have and these are described later. The ORIENTATE mode is operated by pointing the light pen first at the light button "O" then at the component to be moved. The display software returns a pointer to the appropriate segment of the display file. From this the user name may be obtained, which in turn gives a pointer to the component block in the data structure.

A marker cross is displayed at one corner of the component to indicate which one is to be re-orientated. In addition, a small marker arrow appears, pointing in the positive Y direction to indicate the current orientation. By typing C or A on the Teletype keyboard the orientation marker is rotated through 90° in a clockwise or anticlockwise direction respectively. If the new orientation is not allowable for that component, the marker disappears until typing

of further C or A characters brings the marker into an allowable orientation again. When the new orientation has been decided upon, it is implemented by entering the MODIFY mode. Again, the layout algorithm will automatically attempt to fill the current slot to capacity with other components in addition to the re-orientated one.

The restrictions on allowable component orientations are due mainly to the conductor routing subroutines. These will not deal with conductors which have to be routed down one side of the component, across the bottom and out to the other side such as those shown in Fig. 8.3. This is due to the method of component

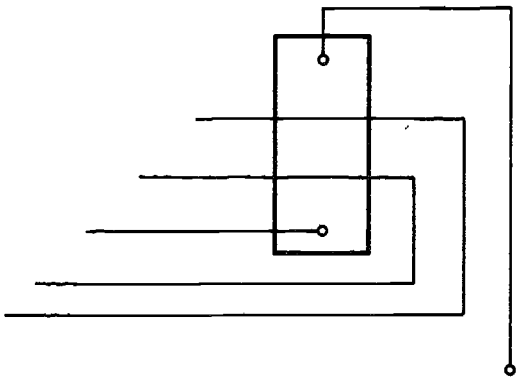


Fig. 8.3 Non-allowable component orientation

orientation discussed in Chapter 7.3.2. Every component therefore has a number of allowable orientations out of a possible four. Branch components have three allowable orientations, the non-allowable one being with the source pin on the upper edge of the component. Subgraph components are only allowed an orientation with the source pin on the lower edge of the component. This normally allows one orientation only. Two are allowable if the source pin is at a corner of the component. During the component orientation subroutine a marker is automatically set to indicate

the allowable orientations. This enables the allowable orientations to be rapidly checked during the interaction subroutine.

8.3.3 PULL Mode

The PULL mode enables the user to pull a component down from a slot to a lower level slot, subject to some restriction. The component to be pulled down is identified by pointing the light pen first at the light button "P" and then at the component. The component is marked by a cross on the display, as shown on component R6 in Fig. 8.1, and a tracking cross appears on the screen. As the tracking cross is moved over the display, a set of three lines indicate the perimeter of the slot in which the cross is positioned. The tracking cross is placed in the slot into which the component is to be pulled. The modification is then implemented by entering the MODIFY mode. When the component is pulled down, one or more other components will necessarily be deleted automatically from the lower slot in order to make room for the new component.

The restriction on pulling down a component is that the base list of the lower slot must contain at least one of the nodes to which the component is connected. The reason for this is that every new component added to the layout is connected to an existing part of the layout. If a component were to be placed in a slot with no connecting base node, there would be no way of knowing which way to route the conductors around the component.

The display of slot boundaries around the tracking cross may also serve a useful purpose prior to the re-orientation of a component. It may be used to check visually whether there is sufficient room in a slot to turn the component. The slot display

is implemented by means of a list of slot dimensions which is built up with the layout. The list may be rapidly scanned and compared with the tracking cross co-ordinates to find the appropriate slot boundaries.

8.3.4 MODIFY Mode

The MODIFY mode is used to initiate the changes required by any one of the above three modes. The purpose of having a separate mode to initiate the modifications is to give the user a safeguard against errors. If he points the light pen at the wrong component by mistake, he can recover from the error before the modification is actually carried out.

Modifications to a slot will alter its placed component profile and will consequently alter the pattern of higher level slots. All parts of the layout above the modified slot must therefore be deleted and later reconstructed with a new set of slots. This is also the reason why only one modification is carried out at a time. If two modifications were to be made in different slots, one slot would probably be at a higher level than the other. As all the layout above the lower slot would be deleted, the modification to the higher slot would then no longer be valid.

Part of the data structure contains a list of all the modifications or changes made to the layout. Each block in the list contains a pointer to a component, the change required and the co-ordinates of the modified slot. The list is ordered in increasing slot level. Each time a change is made, a new block is constructed and inserted into the appropriate place in the list. Any changes in higher level slots are then deleted from the list as they are no

longer valid. The actual implementation of the change is then carried out, described in detail in Section 8.4.

8.3.5 RESET, UNCHANGE and FINISH Modes

The RESET mode is the base state in which the program waits for further control from the user. After a slot has been produced automatically or has been modified, the display is updated and the program returns to RESET mode. The other use of this mode is when the user makes an error in pointing the light pen at a component during PULL, ORIENTATE or DELETE mode. The RESET mode restores the program to its state before the modification was attempted. Marker crosses, slot boundaries, etc. are removed from the display or deleted components are displayed again.

The UNCHANGE mode cycles the layout program automatically through the placement and routing of the next slot. When the slot is completed, the display is updated and the program returns to RESET mode. By repeated entry of the UNCHANGE mode, the whole board layout may be constructed automatically.

It is not always obvious to the user when a layout has been completed. Each time that the UNCHANGE mode is entered therefore, a check is made to see if the layout is complete. If it is, the program enters the FINISH mode. Further modifications may be made if required by returning to RESET mode followed by the required mode. If, however, the light pen is pointed at the FINISH mode light button, the layout is completed and the program is ready for the output of results.

8.4 Modifications to Layout Algorithm for Interaction

The automatic layout algorithm described in Chapter 7 must obviously be modified in order to include interaction facilities. The modifications take the form of four extra subroutines added to the layout program, the basic subroutines remaining substantially unchanged. Two of the interactive subroutines have already been described. These are the display generation subroutine, described in Section 8.2, and the light pen servicing subroutine, described in Section 8.3. The two further subroutines to be described deal with the cutting back of a layout to the level of the latest change and with the actual incorporation of the change into a slot.

8.4.1 Reconstruction of Layout

The base and working lists of the layout algorithm hold detailed information on the state of the layout at the current working level. Once the components have been placed and the conductors routed at this level, the information becomes largely redundant. The redundant base and working blocks are therefore returned to free storage before moving on to the next slot, so as to conserve storage space. It is thus extremely difficult to recall the state of the layout at any level below the current working level. One may find which conductor paths exist at a given level but there is no way of determining to which nodes or conductors they correspond.

The problem of cutting back the layout to the level of a modified slot is approached from a different direction. The entire layout is deleted so that no part remains except for a list of the changes made at the current and lower levels. The layout is then reconstructed, incorporating the changes, up to the level of the

latest change. The layout procedure may then continue from this point. Reconstruction of the layout every time a change is made is not the most economical way of using computer time. This point is further discussed in Chapter 11.4.

The subroutine for cutting back the layout to the level of the latest change starts by adding a new block to the list of changes. The required contents of the block are described in Section 8.3.4. The blocks in the base list, working list, placed component list and other lists of the layout algorithm are all returned to free storage, except for the list of changes. The layout is then reconstructed automatically from the initial base list of edge connector nodes, as described in Chapter 7. Any changes required in the slots are incorporated by the methods to be described below. During the reconstruction, the generation of display file is suppressed. When the currently modified slot is reached, the display is regenerated and the program is ready to proceed under interactive control again.

8.4.2 Insertion of Slot Modifications

The fourth subroutine required for the interaction facilities deals with the actual incorporation of changes into a slot. It operates between the stage of counting and the stage of sorting the contents of the slot, when the layout is being reconstructed. After the total width of all possible contents of the slot have been counted, its co-ordinates are compared with those in the next block of the list of changes. As previously mentioned, each block in the list of changes contains the co-ordinates of a slot to be modified. If there are no changes to be made in the current slot the algorithm

proceeds with the sorting of slot contents as described in Chapter 7.3.4.

When the current slot co-ordinates do coincide with those of the next change block, the component whose position is to be modified is obtained from the change block. The working list is then searched to find the corresponding block. If the component is to be deleted from the slot, its block is deleted from the working list and the total width of slot contents is updated. If the component is to be re-orientated, its orientation and width data are re-computed. Its block is marked to indicate that the orientation must remain unchanged and the total width of the slot contents is updated. If the component is to be pulled down into the slot, its block is marked to indicate that it must remain in the slot. In the case where the user tries to pull a component down into an incorrect slot, the component block will not be found in the working list so a corresponding error message is printed out.

When a modification has been incorporated into the working list, the next block in the change list is examined in case there is more than one modification to be made in the same slot. When all the modifications have been included, the resultant total width of all possible slot contents is compared with the actual width of the slot. Depending on the result, either one of the two subroutines described in Chapter 7.3.4 may be called in order to increase or decrease the width of the potential slot contents.

The subroutine for increasing the width of slot contents is modified slightly so that appropriately marked components are not re-orientated to take up a greater width. The subroutine for

decreasing the width of slot contents is modified so that all the marked components remain in the slot. These measures ensure that the user's modifications are not altered by the automatic part of the layout algorithm. It may occur that the user tries to pull down or re-orientate too many components in a slot. If the width of the slot contents cannot be reduced to less than the actual slot width, an appropriate error message is printed out.

When all the modifications have been made in a slot and when the slot contents have been adjusted to the correct slot width, the layout algorithm proceeds to the placement and routing stage. If then continues automatically, processing each slot in turn and including further modifications where appropriate. The automatic reconstruction of the layout is completed when the currently modified slot is reached. The program is then ready to proceed under further interactive commands.

Chapter 9 Computer Implementation of Layout Algorithm

This chapter describes further programming methods used to implement the layout algorithm. It also indicates the ways in which the programming and form of output of the algorithm have been affected by the available computer hardware and software. The methods of data storage described are extensions of those outlined in Chapter 6.

9.1 Data Structure

The layout algorithm generates and uses many items of data during the construction of a layout. The form and quantity of this data constantly changes as the layout progresses. A data structure is therefore necessary to store the information in the correct order and in a readily accessible form. Much of the data is obtained from the data structure representing the topological model of the layout, described in Chapter 6.

The main additional features required for the layout algorithm are three lists. These are the base list, the working list and the placed component list. An example of one of these lists is shown in Fig. 9.1. During the construction of the layout

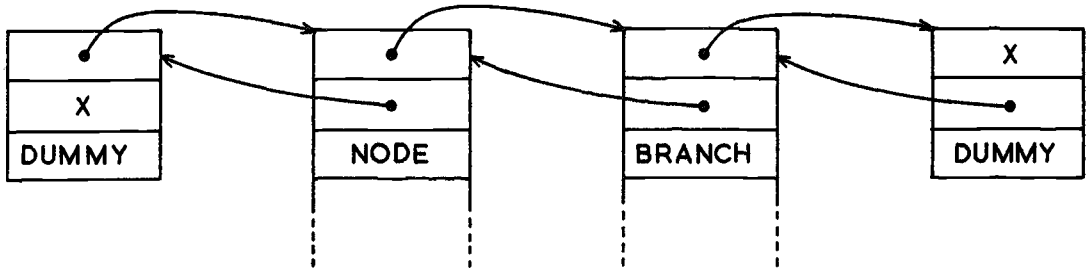


Fig. 9.1 Two-way list used for layout program

it is frequently necessary to know what elements lie on either side of a given component or conductor in one of the lists. The blocks in each list are therefore given two-way pointers as shown in the diagram so that the lists may readily be traversed in either direction.

Every block in the three lists is given a marker to describe the type of element it represents. The base and working lists may contain four different types of element. These are node, conductor, branch component and subgraph blocks. The placed component list contains only placed component blocks. As blocks are constantly being added to and deleted from the lists the problem arises of knowing which blocks represent the ends of the lists. This problem is solved by connecting dummy blocks to the two ends of each list. The same blocks thus remain at each end of the list and when the list is empty, one of its dummy blocks becomes connected directly to the other.

The fourth element of every block in the base and working lists contains a pointer to part of the data structure of the topological model. This enables each block to be uniquely identified. Node, branch and subgraph blocks contain pointers to their corresponding blocks in the topological model. Each conductor block contains a pointer to the tie block in the topological model that represents the corresponding segment of the conductor. The remainder of every block contains data that is obtained and used during the layout algorithm. This includes such items as the total width of a component and its crossing conductors, the source node of a component, the bound branches of a node and so on.

A further type of data structure is required to describe the conductor paths of the physical layout. The structure is illustrated by Fig. 9.2. Each conductor path consists of a

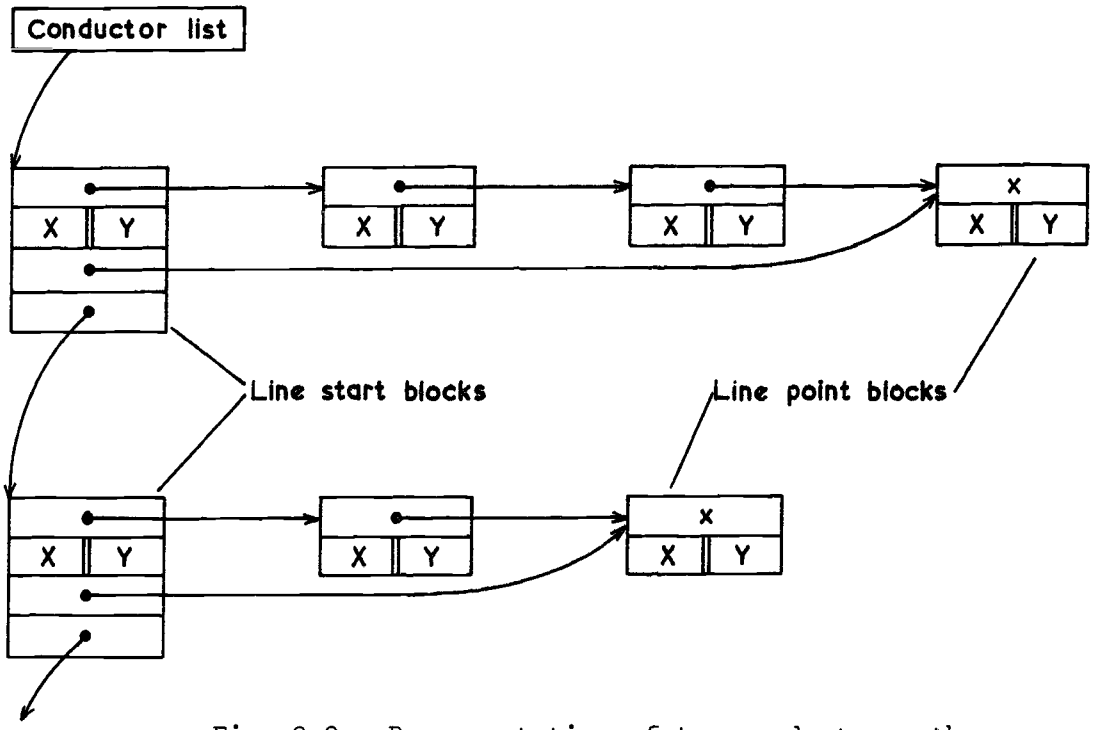


Fig. 9.2 Representation of two conductor paths

line start block together with a number of line point blocks. The line start block contains the starting co-ordinate of the path and a pointer to the list of line point blocks. The line point blocks hold the co-ordinates of the path at each point where it changes direction, including the end point. They also each hold a pointer to the next block of the path.

All the conductor paths of a layout are held in one list. Each line start block therefore contains a pointer to the next block in the list. In addition, each block contains a pointer to the last line point of its conductor path. This enables an extra point to be added to the end of the path without having to

to search through the whole list of line points. It should be noted from the diagram that the two co-ordinates of every block are packed into one word. This saves a considerable amount of storage space in the data structure as a typical layout contains a large number of line point blocks.

If often occurs that a conductor path has to be routed up through several successive slot levels before its destination is reached. It is preferable that one continuous path be defined, rather than have a new line start for each part of the path in successive slots. In order to obtain a continuous path the node or conductor block in the base list which represents the path is given a pointer to the line start block. When the path is routed from the base to the working block, the line start pointer is also passed on to the working block. As this block is later inserted into the base list of a higher slot, the line start is effectively passed up to the next slot level. A further measure is taken to conserve storage space when extending a conductor path. If the path is to be extended in the vertical direction, the Y co-ordinate of the final point is updated rather than create a new line point block.

In programming the layout algorithm, the data structure manipulations are described extensively by use of the ML/1 macro generator. A further set of macro calls are defined and used in the same way as outlined in Chapter 6.3.

9.2 Free Storage System

The data storage system used for the layout algorithm is an extension of that used for the topological algorithm. The data is

stored in a number of blocks which are allocated from one large array, as described in Chapter 6.1. During the working of the layout algorithm however, a large number of data blocks are created and used. When part of the layout has been constructed many of these blocks become redundant. Furthermore, when interaction is used and the layout is modified, most of the layout data structure becomes redundant. A free storage system is therefore added to the basic data storage system so that redundant blocks may be used again by the layout algorithm.

There are a number of ways of arranging a free storage system. Some systems allocate data blocks by dividing up the next largest block. If any blocks returned to free store are adjacent, they are merged into one larger block. Other systems move up all allocated blocks below a returned block so that all the free store is at one end of the data array. Further systems may use a combination of these techniques. The particular system used here is simplified by the fact that the layout algorithm uses only eight different types of block. Every instance of a particular type of block is always of the same length so that there are never more than eight different block lengths in the storage system.

The free storage system is shown in diagrammatic form by Fig. 9.3. The basic part of the system is a store block which contains eight elements, corresponding to the eight different types of data block required. Every data block contains a marker which described its type. These markers are actually the integers from one to eight, so that each type of block may be associated with one element in the store block. All the free blocks of a particular type are thus held in a list. The corresponding element of the

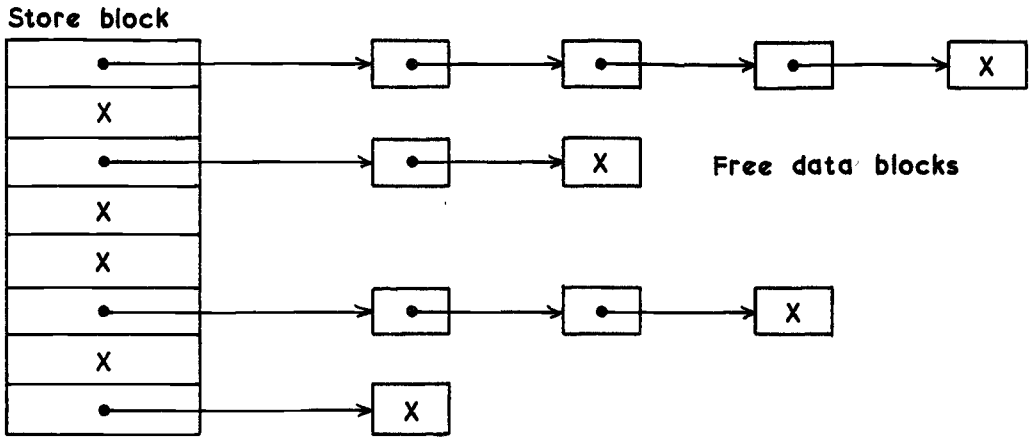


Fig. 9.3 Free storage system

store block then contains a pointer to the start of this list as shown in the diagram.

When a data block of a particular type is required from free store, the corresponding element of the store block is examined. If the element contains a pointer to a list of free blocks the first block is removed from the list and made available to the program. If the element contains a zero-value pointer the blocks of that type have either all been allocated or have not yet been created. In either case a new block is created from the un-used part of the data array in the manner described in Chapter 6.1. To return a block to free storage, its type marker is first obtained. The corresponding element of the store block then indicates the list to which the data block should be added.

9.3 Measurement System of Layout

The system of describing co-ordinates for the layout algorithm is partly influenced by the FORTRAN compiler available for the ICL 4130 computer. One feature of the compiler is that

an array of integer numbers requires one store word per number whereas an array of real numbers requires two words per number. The data array for the layout algorithm is of considerable length so that there is insufficient store space to allow a real array. As the layout co-ordinates are stored in the data array, they must be held in integer form.

Before commencing the layout, a basic unit of measurement is defined by the user. A typical unit could be 0.025". All dimensions and co-ordinates of the layout are then expressed in terms of an integral number of these units. The computer word length is 24 bits which allows the maximum value of an integer to be approximately 8×10^6. This gives more than sufficient resolution for a small basic measurement unit together with a large board size. The reference point of the board from which all co-ordinates are measured is the bottom left hand corner of the board. This assumes that the edge connector lies along the positive X axis.

Every master component in the component library has a reference point and a reference orientation so that its pin co-ordinates may be defined. The reference orientation is such that the longest axis of the component lies parallel to the Y axis, with the reference point at the bottom.left hand corner of the component. The pin co-ordinates may then be defined relative to this reference point. When a component is placed in the layout its position is defined by its orientation and the co-ordinates of its reference point. The relative pin positions are then found by rotation about the component reference point.

The co-ordinates of the conductor paths define the position of the centre line of each path. The minimum distance between centre lines is specified by the user. This distance allows for the width of conductors and the spacing between them. The present version of the layout algorithm permits only one value of conductor width and spacing for the whole layout.

9.4 Output of Board Layout

The basic data describing a board layout is held as a set of integer co-ordinates within the data structure of the computer program. The user generally requires the description of a layout in the form of one or more diagrams showing the placement of components and the routing of conductor paths. The methods of output used at present are described below and some further possibilities of data presentation are discussed in Chapter 11.6.

The display of a board layout is used as the basis of data output for the layout algorithm. The generation of the display file has already been described in Chapter 8.2. The various ways of observing and storing this information are illustrated by Fig. 9.4. The display software (11) also enables the display file to be either punched out on paper tape or transmitted over the high speed link to the PDP-7 computer. Corresponding software in the PDP-7 enables the display file to be read in from paper tape or from the link and then displayed on the Type 340 display.

The display file may be stored from the PDP-7 core onto magnetic tape for subsequent display or plotting. A plotter software package (12) is available to drive the Calcomp plotter from the display file data so as to produce a hard copy of the display.

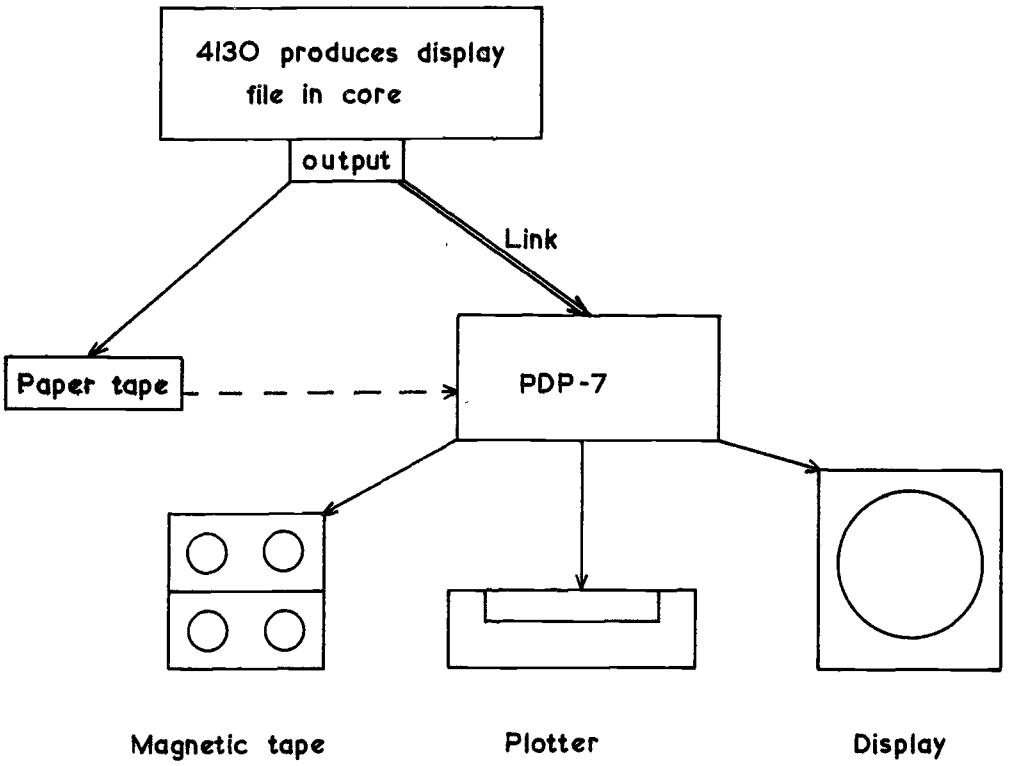


Fig. 9.4 Outputs of board layout program

For present applications the component positions and conductor paths are drawn on the same diagram as shown, for example, in Fig. 8.1. The program may readily be modified so as to produce two separate diagrams of component placement and conductor routing if required.

On completion of a board layout further data is output for the purpose of comparing several different layouts of the same circuit. The total length of all conductors on the board is computed and printed out, together with the overall height of the board actually used by the layout algorithm.

Chapter 10 Results of Layout Procedures

This chapter describes the results of the planarity and layout algorithms. Three circuits are given and their layout diagrams shown. These are then compared with the results obtained by interaction. All the layout diagrams are grouped together at the end of the chapter so that they may readily be compared.

10.1 Description of Circuits

Three different circuits are used to illustrate the results of the layout procedures. They are labelled A, B and C and are shown in Figs. 10.1, 10.2, and 10.3 respectively. They are typical of the smaller type of industrial circuits that are laid out on single sided boards.

Circuit A has been used for most of the development and testing of the algorithms so detailed data is available for all stages of its layout construction. The circuit is used to show the results of the planar graph and pseudo-planar graph algorithms. The circuit layouts also illustrate the improvements that may be made by the use of interaction.

Circuit B, of similar size to A, is again used to illustrate the layouts obtainable by automatic and interactive means. In addition, a manually-designed layout of this circuit is given. The computer and the manually generated layouts are compared and the different techniques discussed.

Circuit C has approximately twice the number of components of the previous two circuits. It is used to illustrate the effects on computer time and storage space of larger circuits. It also shows the improvements that are possible by the use of interaction.

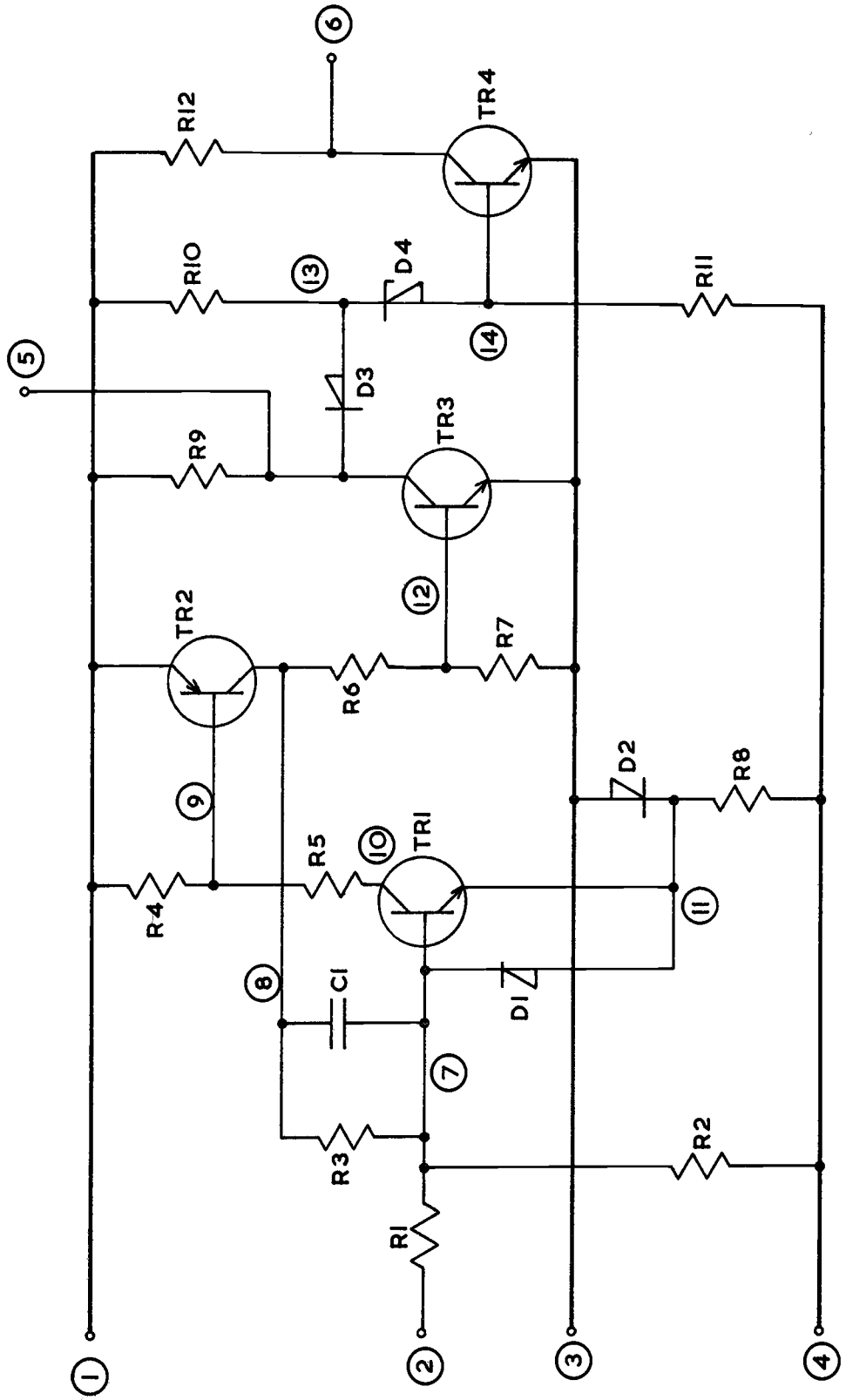


Fig. 10.1 Circuit diagram of circuit A

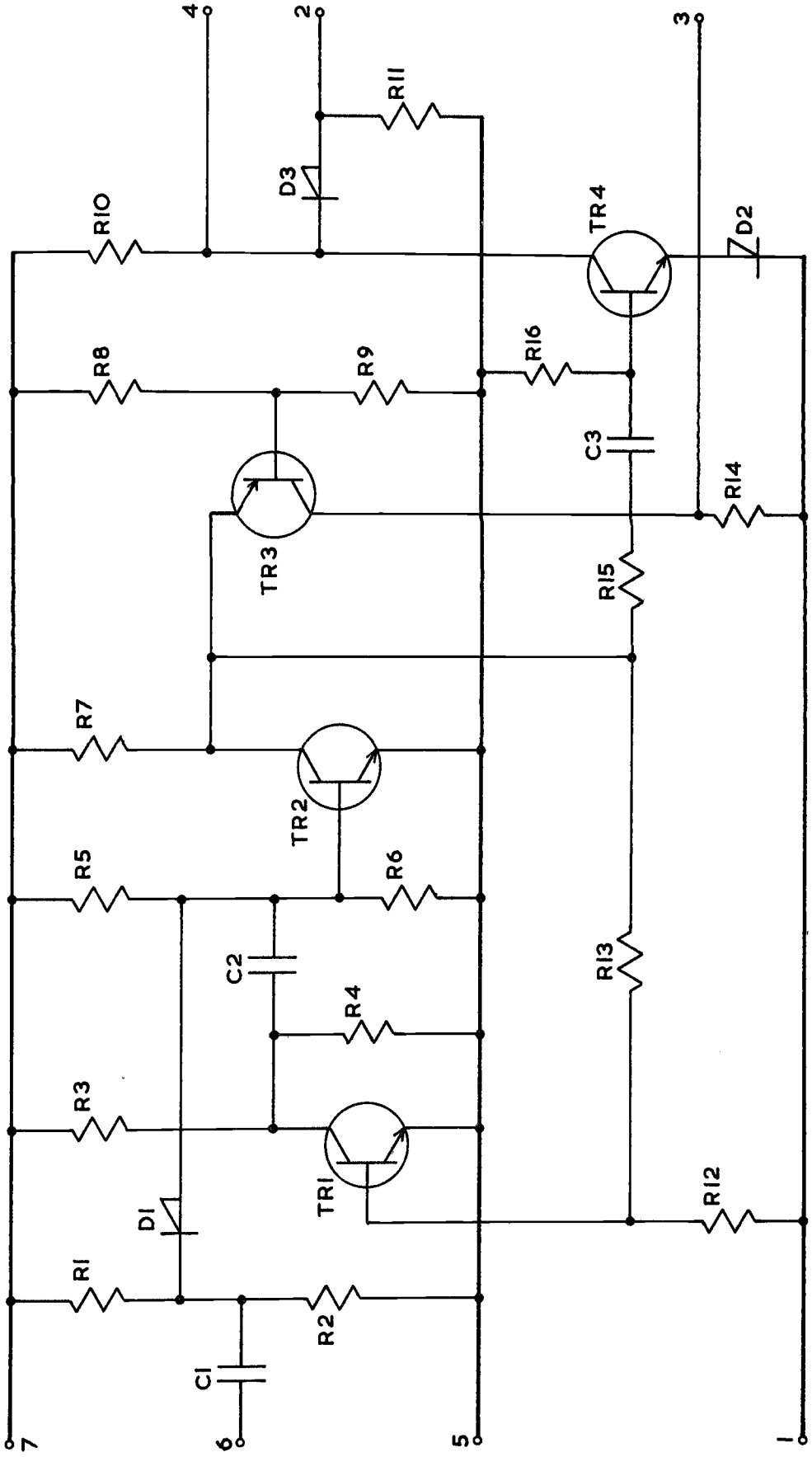


Fig. 10.2 Circuit diagram of circuit B

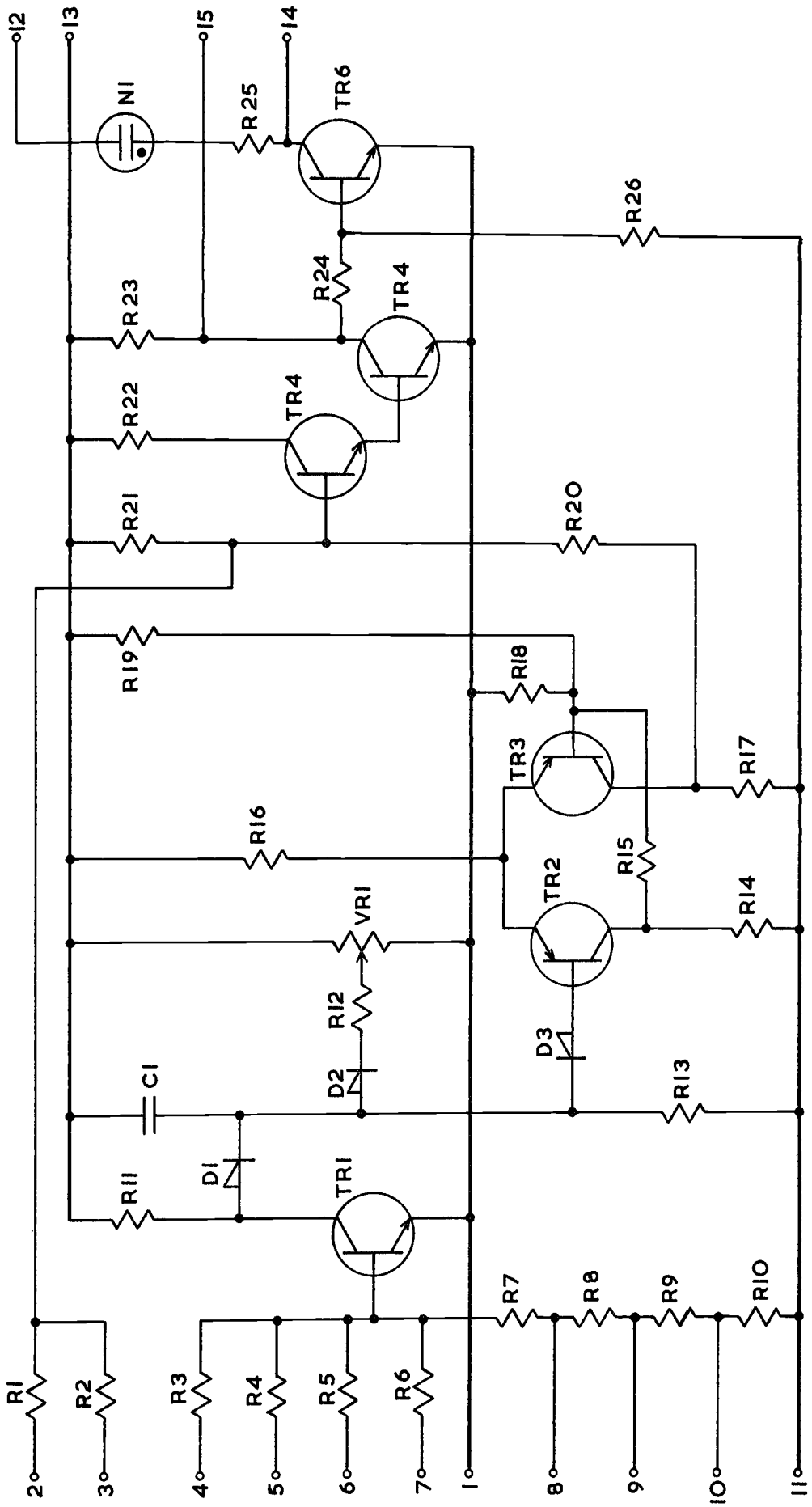


Fig. 10.3 Circuit diagram of circuit C

10.2 Construction of Pseudo-Planar Graph

The construction of the pseudo-planar graph of a circuit is an essential part of the layout procedure. As the planarity algorithm is completely automatic, and the results are then used in the generation of the physical layout, the graph is not normally output from the computer. When output of the graph is required, it is printed out in the form of a list of regions. Each region is itself a list of the branch segments which form the boundary of the region. Although this form of data is ideally suited to the planarity algorithm, it results in a difficult task when constructing a diagram of the graph. The comparison of planar and pseudo-planar graphs has therefore been made for one circuit only.

10.2.1 Construction of Planar Graph

The initial planar graph of circuit A is shown in Fig. 10.4. The circled numbers are the circuit nodes, corresponding to those labelled in Fig. 10.1. Subgraph nodes are labelled by their transistor number followed by a letter A, B or C denoting the collector, base or emitter of the transistor respectively. Component branches are labelled with their appropriate component name. The branches shown dotted are those which have been removed from the total graph in order to make it planar. The outside edge of the graph is composed of the edge pseudo branches and the edge connector nodes, labelled from 1 to 6. The first region of the graph is then that which lies outside the boundary of the graph.

The first starting node taken in the construction of the planar graph is node 1 and the first target is node 2. The search for a planar path between these two nodes yields the components R1, C1 and TR2. These components, together with the edge pseudo branch

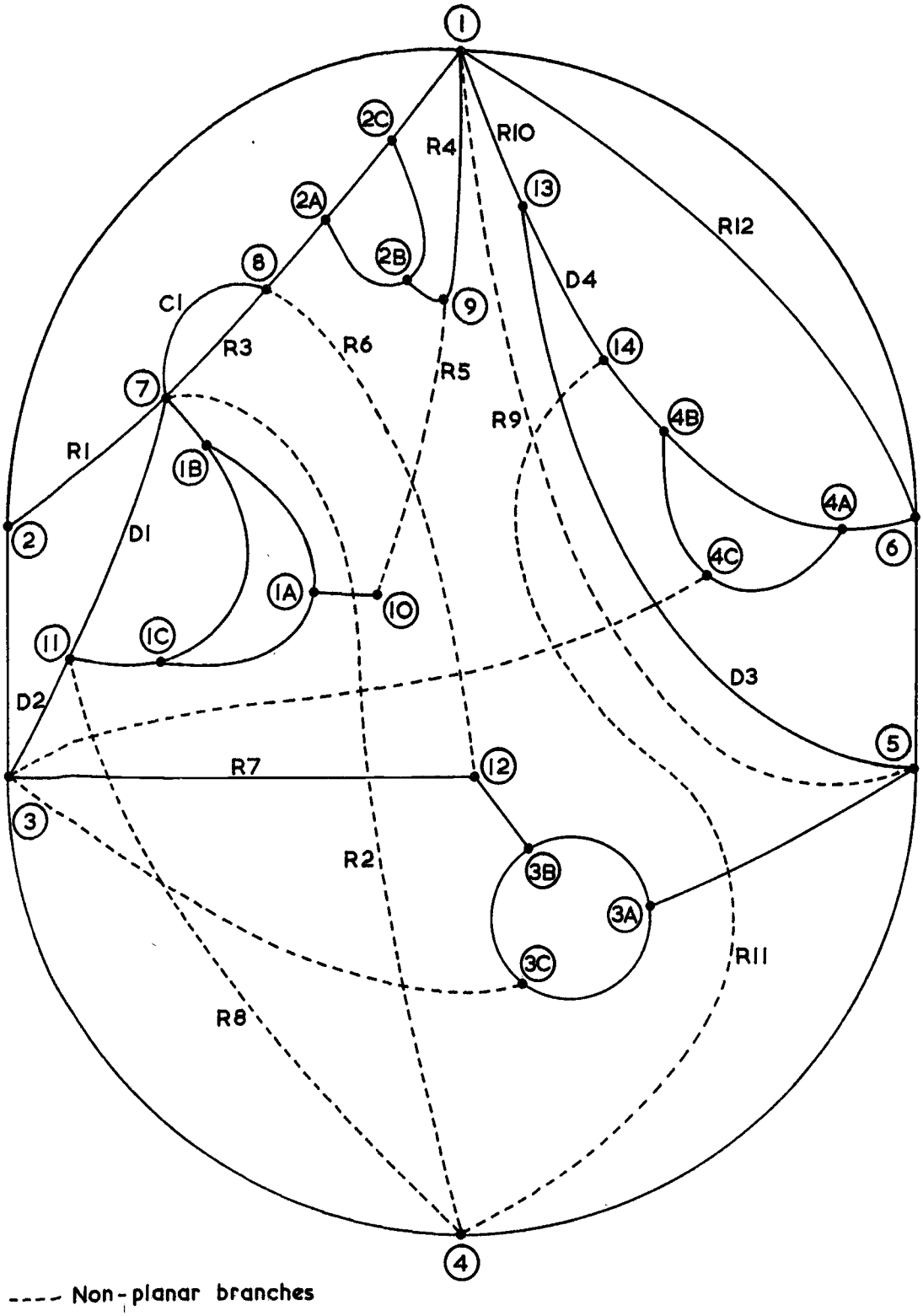


Fig. 10.4 Planar graph of circuit A

1-2 therefore form the boundary of the second planar region. The remaining two pseudo branches of subgraph component TR2 are then added to the graph as a further planar region. The next target in an anticlockwise direction from the start node is thus node 2B. Another planar path is then found, adding component R4 to the graph. The following target is node 9. As no planar path exists between this node and the starting node 1, the search direction is changed to a clockwise direction from the start node.

Two further planar regions are added to the graph by searching for planar paths from node 1 to node 6. The following target is then node 13. No path exists from node 1 to node 13, however, without touching the edge of the free region at some other point and hence dividing the free region into two parts. The remaining branch on the start node, branch R9, is thus deemed to be non-planar as it conflicts with component branch R11 and link branch 4C. The algorithm continues by taking further nodes in turn as starting nodes. These nodes are 9, 8, 7, 1A, 11, 3, 4 and 5 in that order. The planar graph is then complete as shown in the diagram.

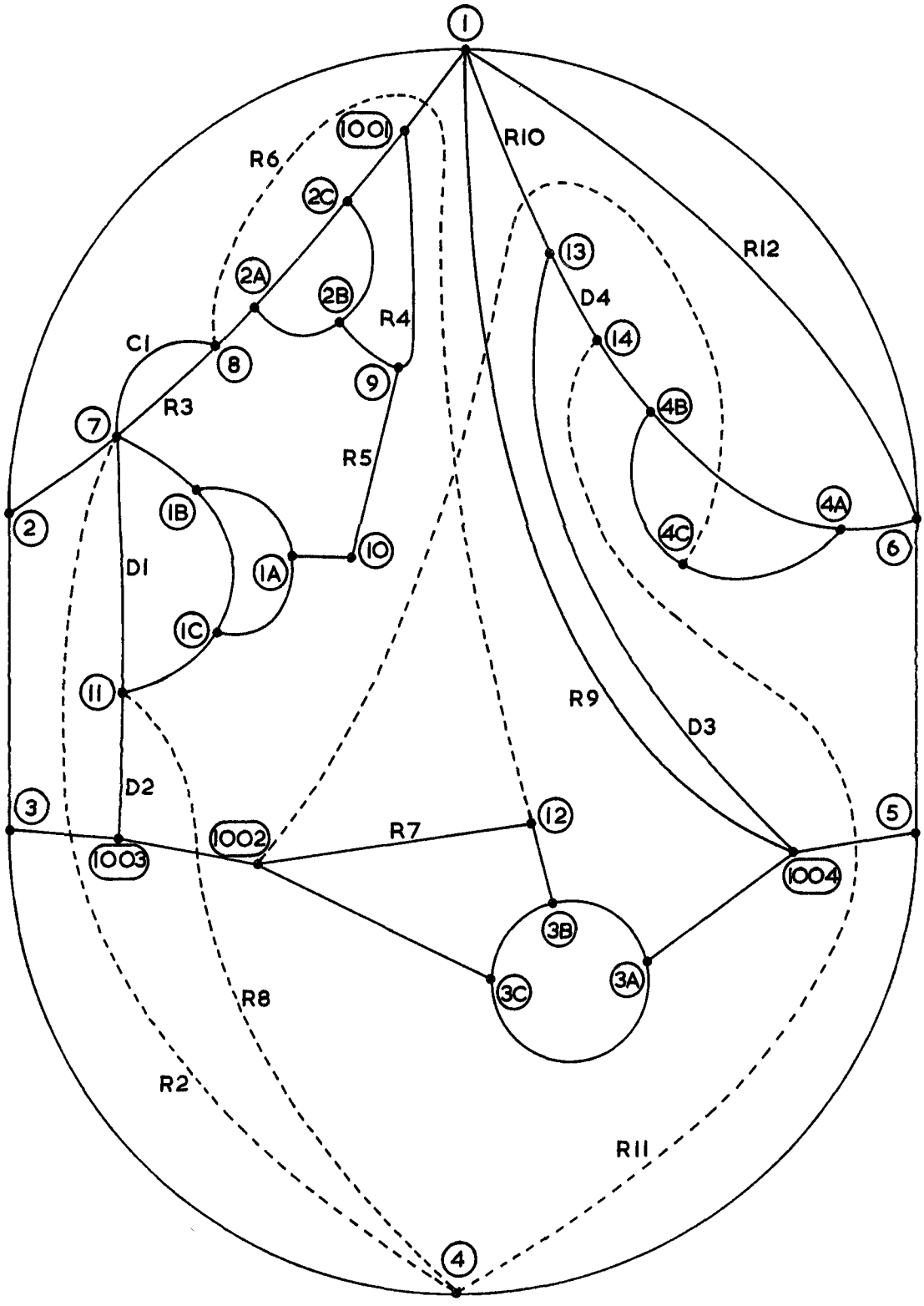
It can be seen from the planar graph that the two branches with which R9 conflicted, R11 and link 4C, have also been removed from the graph at a later stage. Branch R9 could thus be included in the planar graph. Similarly there are two other branches which could be included. These are link branch 3C and either component branch R5 or R6. The planarity algorithm thus does not necessarily select the optimum planar subset of a graph. This is not critical however as the planar branches removed from the graph are recognised and re-inserted during the next stage of the algorithm.

The insertion of component branch R12 into the graph illustrates a difficulty in the construction of a physical layout from a topological model. The branch constitutes the shortest possible path, or path of minimum number of branches, between nodes 1 and 6. The edge pseudo branch 1-6 however, represents the outside edge of the board. To connect component R12 into the layout therefore, its connecting conductors must be routed around three sides of the board. This illustrates the problem that the shortest distance in the graph does not necessarily represent the shortest distance in the layout.

10.2.2 Insertion of Non-Planar Branches

The completed pseudo planar graph of circuit A is shown in Fig. 10.5. It is substantially the same as Fig. 10.4 except that the non-planar branches have now been assigned fixed paths in the graph. These branches are still shown as dotted lines so that they may readily be recognised. The nodes labelled with numbers greater than 1000 are new nodes formed by the "node splitting" process of inserting non-planar component branches.

The effects of the "node splitting" algorithm can clearly be seen in the diagram. Part of node 1, for example, is split into node 1001 so that component branch R6 may be inserted into the graph. Part of node 3 is split into node 1002 for the insertion of branch R8. It is then split again into node 1003 for the insertion of R2. The number of parts into which a node may be split is limited only by the number of branches connected to it. It may also be observed that the three planar branches mentioned above have been recognised and inserted into the graph.



---- Non-planar branches inserted

Fig. 10.5 Pseudo-planar graph of circuit A

A further point to note from the diagram is that non-planar branches R6 and link 4C are crossed. Branch R6, being a component branch, is inserted into the graph first. Although it crosses one branch in splitting node 1 there is still space for further crossings under the component. When the link branch is later inserted, the insertion algorithm is concerned only with the amount of space under component and pseudo branches already in the graph. Hence a non-planar component branch may later be crossed by several non-planar link branches.

10.2.3 Comparison of Circuits

The relative sizes, storage requirements and computing times of the three circuits used are compared in Table 1 below. The computing time given is the approximate time required to read in the data, set up the data structure and construct the pseudo-planar graph of the circuit. The storage requirement is the number of words of the data array used in the construction of the pseudo-planar graph.

Circuit	A	B	C
No. of components	21	26	38
No. of circuit nodes	14	16	32
Storage (words)	1345	1731	3157
Computing time (secs.)	5	5	11

Table 1 Comparison of Circuit Sizes and Computing Times

A further point arises from the construction of a pseudo-planar graph. If the algorithm is started from another node on the edge connector it generally produces a different graph of the same

circuit. This occurs naturally as no attempt is made to search for the optimum planar graph of the circuit. The computing time required to generate a pseudo-planar graph is short, as can be seen from Table 1. It is therefore feasible to start at a different node and generate a different graph of the circuit if, for any reason, the first graph is unsatisfactory.

10.3 Construction of Layouts

The layouts constructed automatically by the layout algorithm, with no alterations by the user, are discussed here. The results of interaction are described in the next section.

10.3.1 Layout of Circuit A

The layout of circuit A is shown in Fig. 10.6 and clearly illustrates a number of features of the layout algorithm. The packing density of components on the lower part of the board is good and conductor lengths are short. This is due to the fact that the first components selected for placement are those closely connected to the edge connector. The next components selected are then those most closely connected to the existing part of the layout. This strategy produces a compact layout as intended.

The upper part of the board has a lower component density and contains a number of long parallel conductor paths. This is mainly due to connecting up ends of node and conductor paths which have already been started at lower levels of the layout. When processing base nodes in the higher slots, they frequently develop into components which have already been placed on the board or into conductors which have to cross under these components. The upper part of the board therefore contains a higher ratio of conductors

to components in order to preserve the circuit topology. This weakness of the algorithm can, however, be improved by the use of interaction.

The circuit contains one extremely long conductor. This is the link branch which connects component TR4 to TR3 and R7 and which crosses under R10, R9 and R6. This length of conductor path is undesirable in a practical layout because it increases the board space required for routing and may introduce excessive stray capacitance between adjacent conductors. It illustrates the fact that a short path in the graph does not necessarily represent a short path in the layout. The path is necessary in the layout, however, in order to preserve the circuit topology. A draughtsman laying out the circuit would either re-arrange the component positions or insert a wire jumper in order to reduce the conductor length.

Part of the board space is un-used in the slot bounded by the board edge and component C1, and above components R10, R9 and R1. Development of the base nodes of this slot yields either components which have already been placed or conductors which are to cross under other components. No components can therefore be placed in the slot so the available space is wasted. It is obvious from the diagram though, that component TR2 could be placed in this slot even though its source node lies outside the slot boundary. An extension of the principle of the PULL mode of interaction could thus be used to automatically pull components down into empty slots and hence improve the component packing density.

The crossing conductor between components R10 and R9 is the cause of frequent comment. The actual component-crossing parts of

the conductor are inserted at the same time as the components. At a later slot level (across the tops of the components) the conductors are routed up to that level before they are recognised as two parts of the same conductor and joined. The same procedure is absolutely necessary however in the case of the conductor connecting R11 and TR4. If the two parts of this conductor are not routed to a level above TR4, the conductor will clash with the crossing under TR4. Unnecessary bends in conductors such as that between R10 and R9 could be avoided by further programming. This would check for the absence of components between the two parts of the conductor before routing the conductor path.

10.3.2 Layout of Circuit B

The automatically produced layout of circuit B is shown in Fig. 10.8. Most of the observations on circuit A also apply to this circuit. One point that is immediately obvious is that the layout has "fallen off" the top edge of the board. Although such a layout could not be built it shows a useful property of the layout algorithm. The algorithm will continue over the edge of the board and still show the state of the layout. It will not, as some layout programs do, go into an error state when there are too many components to fit onto the board.

There are two possible courses of action when the layout exceeds the board size. If the layout is mostly on the board it may probably be arranged wholly on the board by the use of interaction. If a large proportion of the layout is off the board, the required component density is too high. The circuit must therefore be placed on a larger board or partitioned onto two separate boards.

Unlike the previous layout, circuit B is laid out on a board whose width is less than its length. A considerable area of the board space is thus taken up by conductors routed up to components at higher slot levels. In the limiting case the board width would be almost entirely taken up by conductors and there would be no room for further components to be placed. This situation can be partially remedied by interaction and can be eliminated by using a wider board.

The layout algorithm optimises the contents of each slot in turn. This may not however give the optimum overall layout as is illustrated by component R2 at the top of the layout. R2 is placed in the slot across the top of R14 and bounded by the edge of the board and component R16. There is only sufficient room in this slot to place R2 in a vertical orientation. If placement had been delayed to the later slot across the top of R16, the component could have been horizontally orientated. This would then have reduced the overall height of the layout. This is a typical case where interaction can be used to improve a layout.

10.3.3 Layout of Circuit C

The automatically produced layout of circuit C is shown in Fig. 10.11 and illustrates the layout of a larger circuit. It can be seen that the component packing density on the left hand side of the board is good. The components are closely interconnected with few crossing conductors. In comparison the right hand side of the board is largely taken up by a number of parallel conductor paths.

Closer examination of the layout reveals that there are three or four conductors which follow parallel paths under components VR1, R14, R19, R22, R26, R23 and R25 in that order. These are the

conductors which occupy most of the right hand side of the board. The paths are necessary in order to preserve the circuit topology and they once again demonstrate the problem involved in translating a topological model into a physical layout. The excessive space requirements may be reduced by the use of interaction. In the case of a manually produced layout, the conductor paths would probably be avoided by the use of wire jumpers.

The diagram shows that the spacing between adjacent components with crossing conductors is greater than necessary. Examples of this are components R1, R2 and R3 on the left and components R25 and R23 on the right hand side of the bottom slot. The reason for the unnecessarily long crossing conductor paths has already been explained for circuit A. If the lengths of these paths are reduced by further programmed checks as suggested, the same information can be used to reduce the spacing between the adjacent components. This would then improve the component packing density of the layout.

The diagram shows that the orthogonal routing of conductors could be improved in some cases. For example, the conductor paths from the top and right hand sides of R14 could each have several change points removed by routing the conductors vertically as far as possible then horizontally. Storage space for the conductor paths would be reduced also. Conversely, the same treatment could not be applied to the conductors below components R8, R17, R13 and R14. The conductor routing can thus be improved at the cost of further computational checks during routing or by allowing interaction with the conductor paths.

10.3.4 Comparison of Computer Requirements

The computing times and storage requirements for the three layouts are compared in Table 2 below. To obtain the computing times, the generation of display and the interaction subroutines have been suppressed. This has been done to eliminate the user interaction time and the time taken to completely regenerate and transmit the display file for every slot. The time given is thus that required to automatically generate the complete layout from an existing topological data structure. It does not include the time taken for the output of results as this is dependent on the form of output used.

The storage requirement for each circuit is the number of words of the data array used by the layout algorithm. This comprises storage for the topological model, the layout, its conductor paths and the data blocks used for the base, working and other lists.

Circuit	A	B	C
Computing time (secs.)	8	13	26
Storage space (words)	2995	4258	8243

Table 2 Comparison of computing times and storage space

10.4 Results of Interaction

In this section the results of interaction with the three board layouts are discussed and compared with the automatically produced layouts. It should be emphasised that the modified layouts are not unique. A completely different, possibly better, layout may be obtained for each circuit by carrying out different modifications.

10.4.1 Interaction With Circuit A

The modified layout of circuit A is shown in Fig. 10.7. Comparing it with the automatic layout of Fig. 10.6, it can be seen that the component packing density has been improved. A number of long parallel conductor paths have also been eliminated from the layout.

The basic strategy of interaction in this case is the observation that six of the parallel conductors across the top of the automatic layout are developed from the edge pin 1 and components R9 and R10. The two components are themselves developed from edge pin 1. If the components were on the right hand side of the board, only one conductor from edge pin 1 would have to be routed across the top of the layout. The other five conductor paths would then be drastically reduced in length.

To produce the modified layout, components R9 and R10 are deleted from the first slot. The layout algorithm compensates for the change in slot contents by automatically re-orientating the components in the bottom slot as shown. Continuing with the layout algorithm, the next slot to be processed is that across the top of R12 and bounded by TR4 and the edge of the board. It so happens that R9 is placed in this slot by the program. The user then continues to use the automatic facility of the algorithm to produce the remainder of the layout.

From the layout of Fig. 10.6 it can be seen that some conductor paths could be shortened by rotating component TR4 through 90° anticlockwise. The automatic algorithm has not done this because TR4 has a choice of two possible orientations. The information available at the time of assigning the orientations is

not sufficient to choose the better of the two. The modification has therefore been made by the use of interaction as shown in Fig. 10.7.

Although this circuit gives a simplified example of interaction, it illustrates how significant improvements can be made to a layout by a few modifications in conjunction with the automatic algorithm. The user's ability to look ahead from a slot to later parts of the layout enables the overall layout to be optimised, rather than the contents of each slot. Actual figures on the improvements to the layout are given in Table 3 below.

Due to the increased packing density of the layout after interaction, a large blank space is left at the top of the board. There are a number of ways of dealing with this, depending on the user's requirements. The layout may readily be expanded in the Y direction so as to fill the whole board space. Alternatively it may be left as it is, or re-laid out on a smaller board.

10.4.2 Interaction With Circuit B

The modified layout of circuit B is shown in Fig. 10.9 and may be compared with the automatic layout of Fig. 10.8. It can be seen that interaction has reduced the layout size to bring it well within the bounds of the board. Two basic interaction strategies are used for this layout. The first, as for circuit A, involves recognising that some components developed from a base node produce a number of conductors which are routed up to higher slot levels. These components, such as TR4 in the bottom slot of Fig. 10.8, are thus moved up to a higher level so as to reduce their connected conductor lengths.

Removal of TR4 from the bottom slot leaves room for further components in the slot (D3, C1 and R1) and consequently gives a greater packing density. The layout is continued and components such as TR4 are moved upwards to higher levels until most of their nodes can be connected to adjacent conductors. The moving of a component up to a higher level consists of deleting it from all the slots it appears in until the required level is reached. This is sometimes tedious and could possibly be improved by having a further mode of interaction to pull components upwards.

The second interaction strategy involves arranging the desired orientations of components. An example is given by components R13 and R15 in Fig. 10.9. Previously the components were orientated vertically in the slot across the top of R8 and TR2, and bounded by C3 and R2. This gave a greater height to the layout and left spare space in higher slots to the left of the components. The two components are re-arranged by deleting R13 from the slot. This leaves sufficient space for R15 to be orientated horizontally. R13 is then orientated horizontally in a higher slot. The same technique is also used to move component C2 from the right hand side of the board to the top of the layout.

One important point noted during interaction is that modifications to a layout should be made at lower levels first. If this is not done, a later modification at a low level will delete the layout and modifications above that level. In some cases, such as pulling a component down to a lower level, this is unavoidable and means that the higher level changes will have to be made again.

10.4.3 Interaction With Circuit C

The modified layout of circuit C is shown in Fig. 10.12. Comparing it with the automatic layout of Fig. 10.11 it can be seen that interaction has made a considerable improvement to the layout. The techniques used to improve the layout are similar to those described for the previous two circuits. An additional strategy has also been used, based on the observation mentioned in section 10.3.3 that a number of components are crossed by the same three or four conductors. The lengths of these conductors can be considerably reduced by arranging that the crossed components are adjacent to each other. The results of this strategy can be seen in the centre and right hand side of the layout.

At higher levels of working on the layout, the time taken to make a modification becomes quite noticeable. This is due to the fact that the layout and the display are completely regenerated up to the level of the modification. Possible improvements to this situation are discussed in Chapter 11.4. The modified layout still contains a number of long parallel conductor paths. It is possible that with further interaction some of these paths may be reduced in length. It is a general point, however, that the improvements to a layout are ultimately limited by the circuit topology.

10.4.4 Comparison of Interaction Results

The "goodness" of a layout is difficult to specify. It depends partly on the overall appearance of the layout and partly on the user's special requirements. Often, two completely different layouts of the same circuit may be equally satisfactory. For this reason two simple criteria are used for comparing layouts generated

with and without interaction. These are the total conductor length of the layout and the overall height of the board used by the layout algorithm. The comparison of circuits is made in Table 3 below.

Circuit	A	B	C
Conductor length without interaction	2446	5548	11802
Conductor length with interaction	1185	3714	6972
Percentage reduction in length	51%	33%	41%
Layout height without interaction	82	174	152
Layout height with interaction	62	130	114
Percentage reduction in height	24%	25%	25%

Table 3 Improvement of layouts with interaction

The storage requirement of each of the three circuits is approximately the same as that given in Table 2. The storage space used cannot be measured accurately as it is dependent on the amount of interaction carried out to generate the layout.

10.5 Comparison With Manually-Generated Layout

The modified layout of circuit B shown in Fig. 10.9 is compared with a manually-generated layout of the same circuit, shown in Fig. 10.10. The main difference between the two is that the manual layout makes far more use of the space under components for conductor routing. This is in contrast to the topological method which uses component crossings only as a last resort when inserting branches into the graph. The consequent results are that the computer layout requires a larger board area with a greater proportion of the space taken up by conductor paths.

A further difference is that the manual layout has all the components placed in the same orientation. This is usually done to assist the manufacture of the board and to give it a more pleasing appearance. The computer layout tends to pack the components in a number of different orientations so as to make better use of the available board space. A similarity between the two layouts is the number of parallel conductors across the top of the board. Both approaches have similar problems of preserving the planarity of conductor paths on a single-sided board.

The overwhelming advantage of the computer method is the time taken to produce the layout. A draughtsman would take several hours to produce the completed layout diagram. The computer method takes minutes to produce an initial layout with perhaps half an hour of interaction time to improve the layout. In addition, the output of the program may be used to drive a mechanical plotter to produce the finished drawings of the required accuracy. For a large layout the corresponding saving in time can be several weeks.

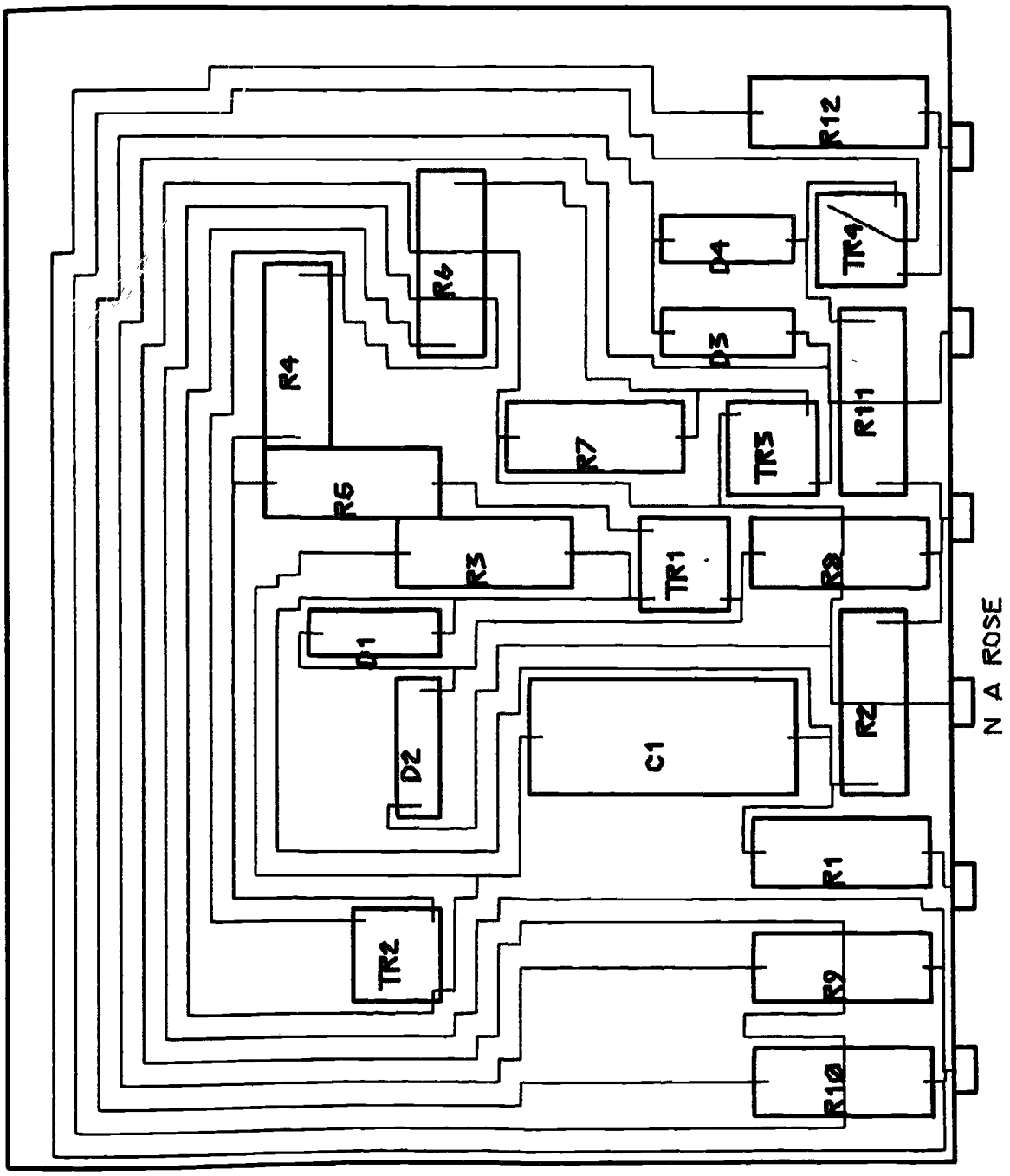


Fig. 10.6 Automatic layout of circuit A

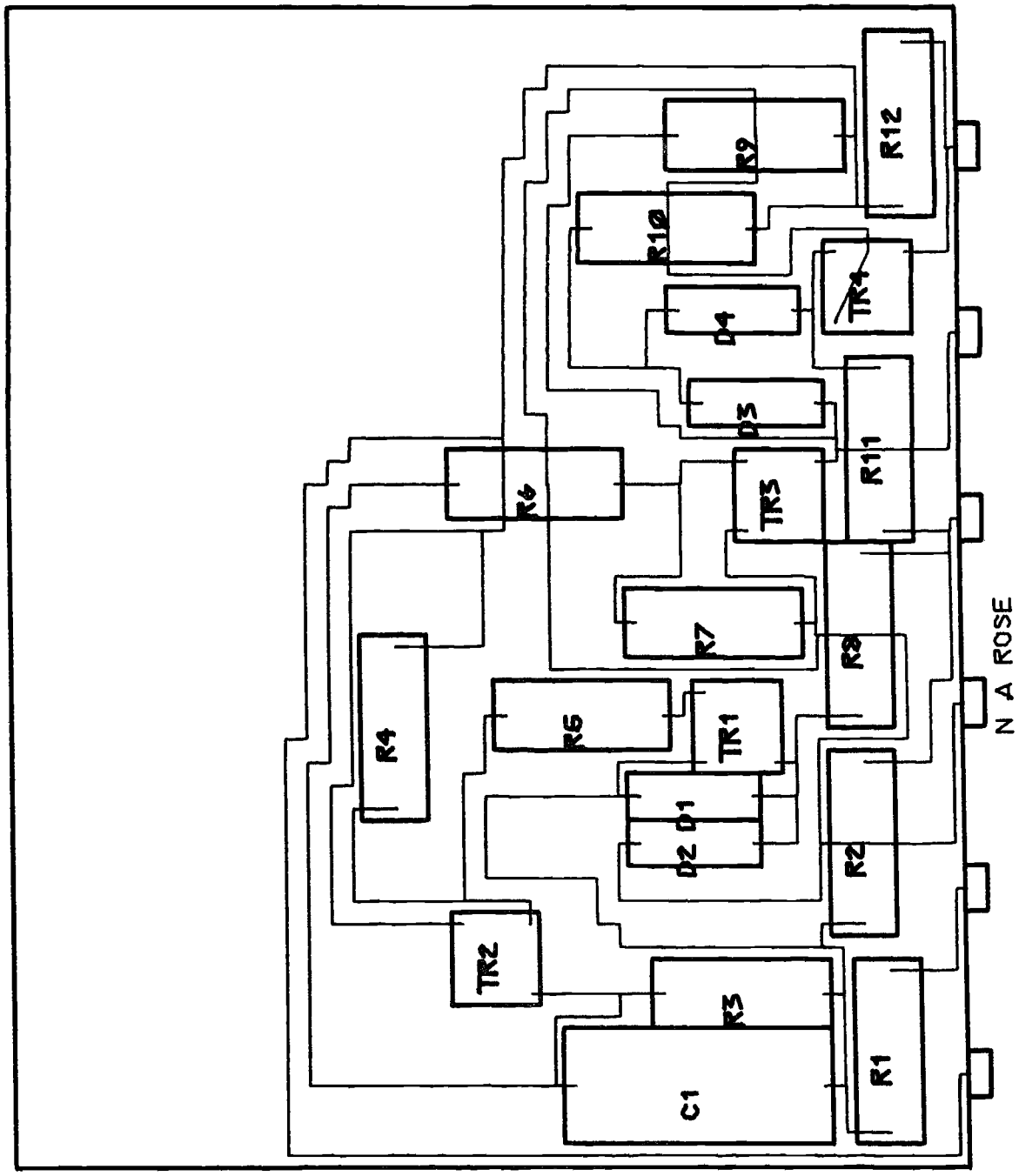


Fig. 10.7 Layout of circuit A modified by interaction

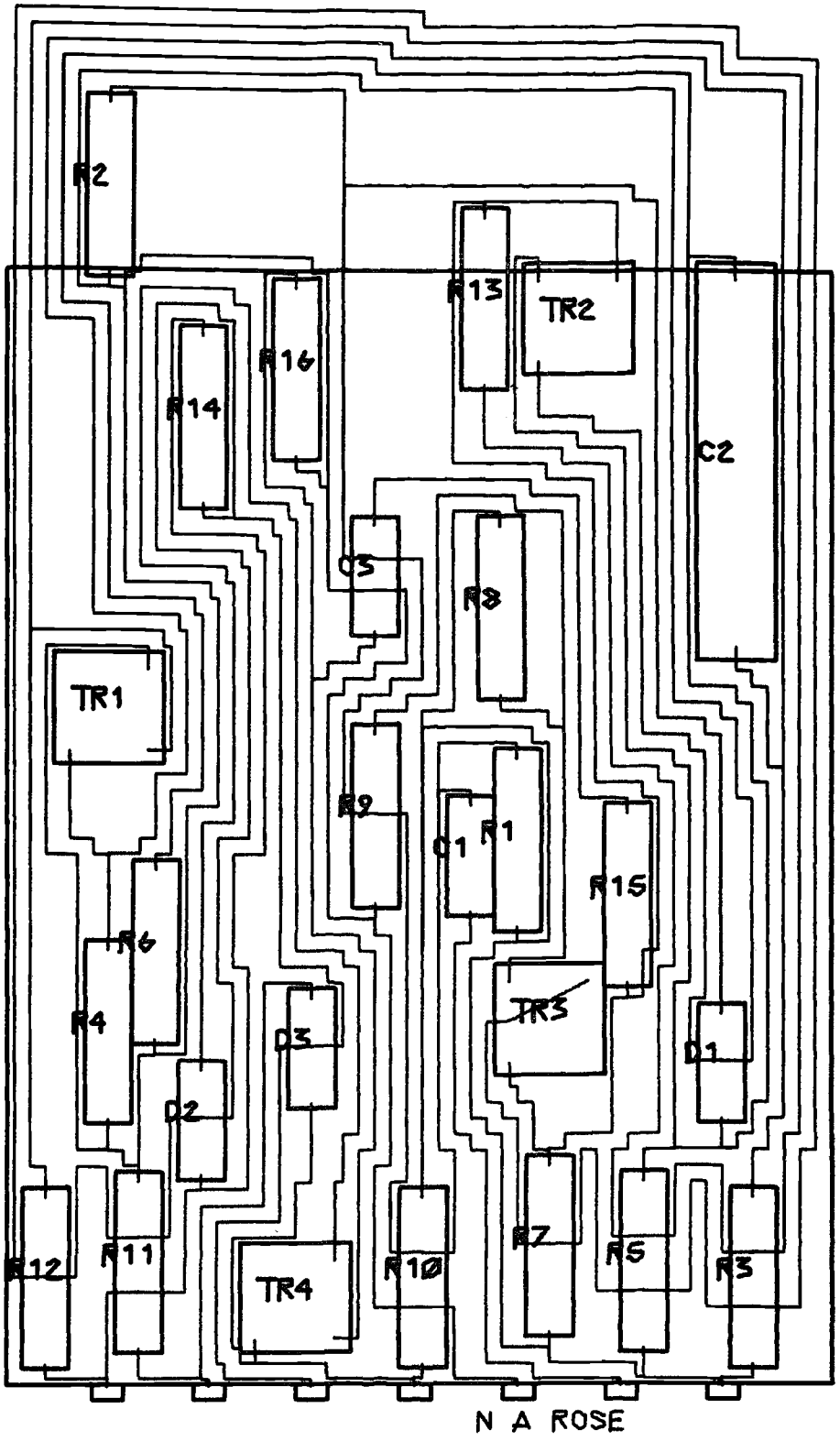


Fig. 10.8 Automatic layout of circuit B

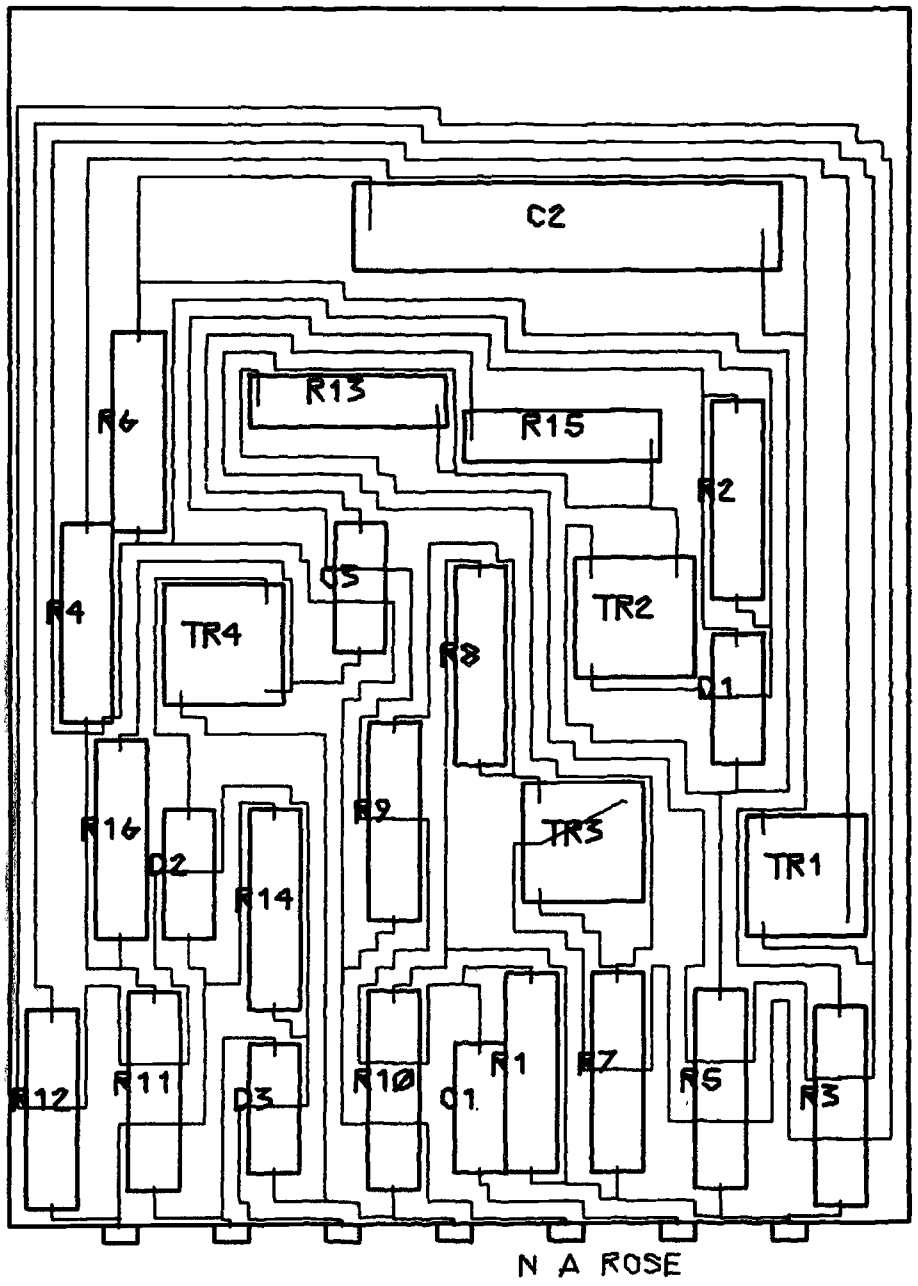


Fig. 10.9 Layout of circuit B modified by interaction

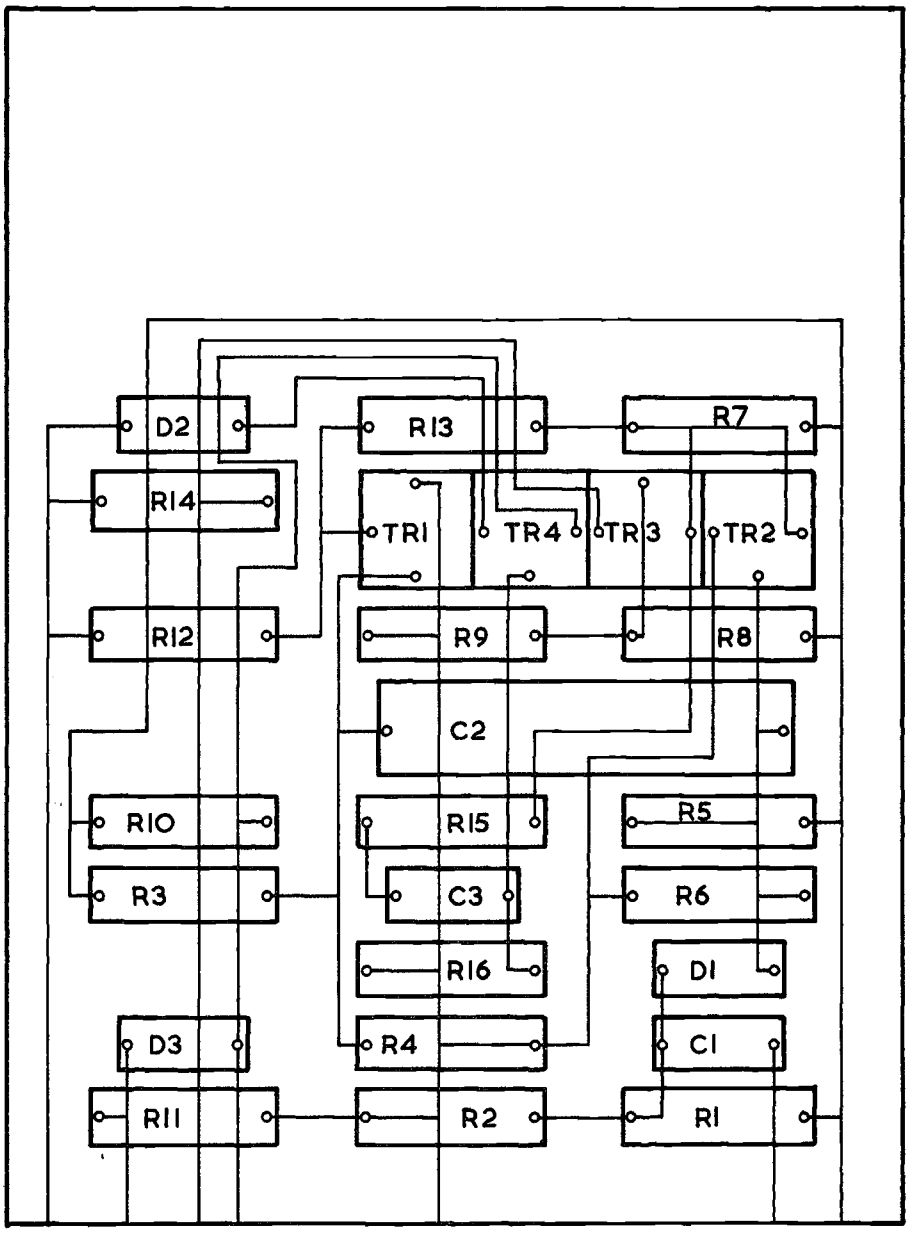


Fig. 10.10 Manually-produced layout of circuit B

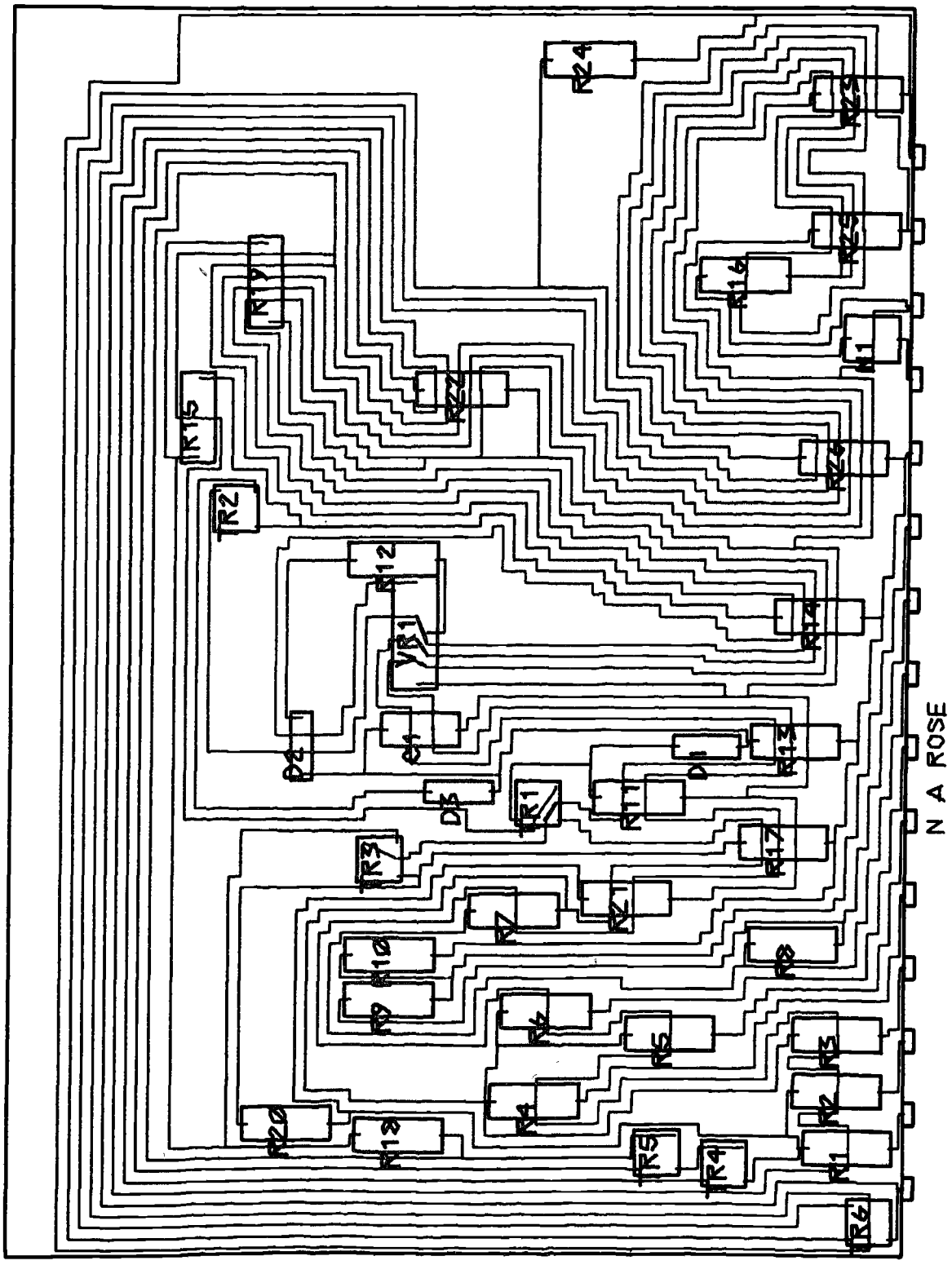


Fig. 10.11 Automatic layout of circuit C

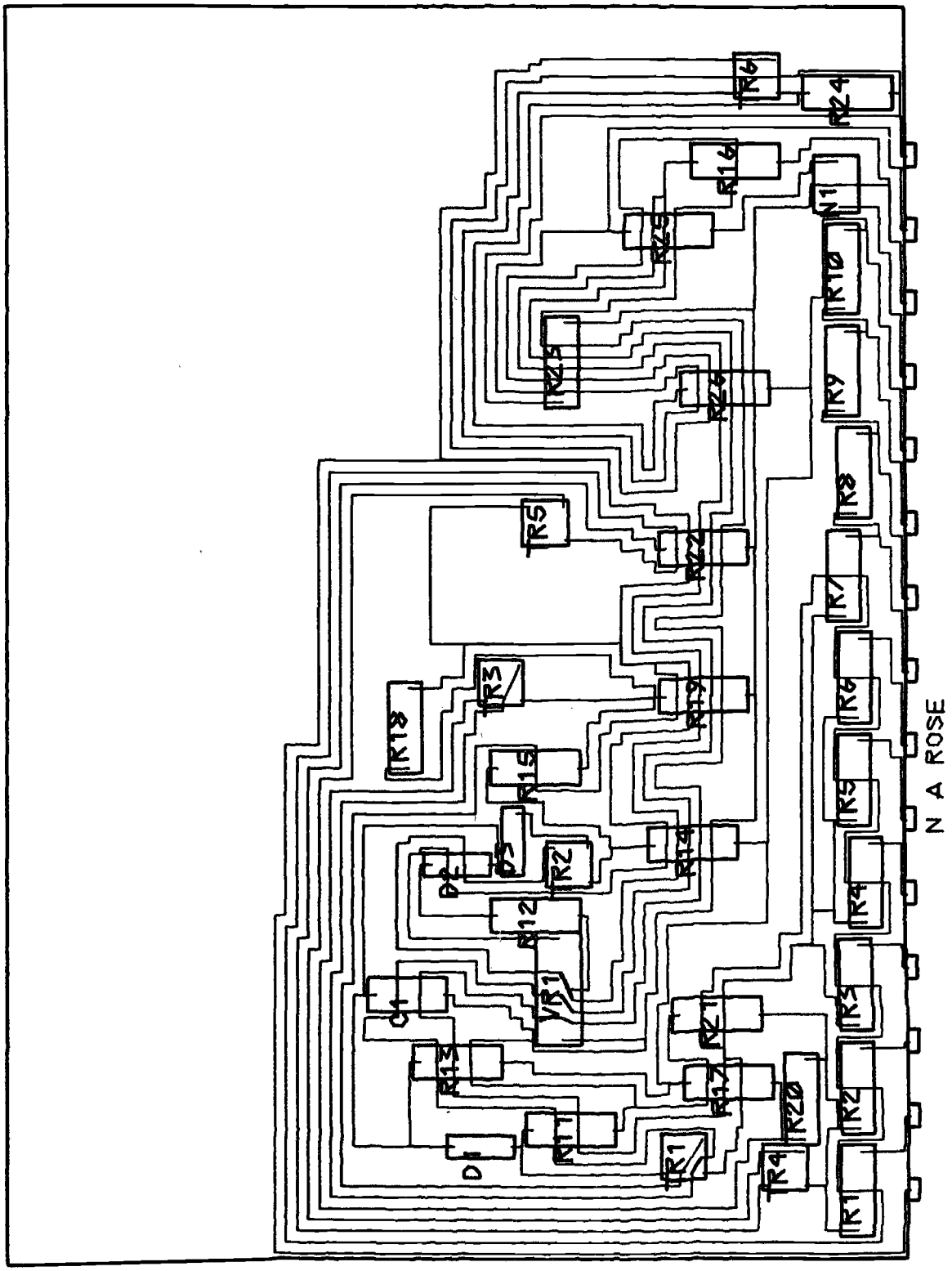


Fig. 10.12 Layout of circuit C modified by interaction

Chapter 11 Discussion of Method and Improvements

It is clear from the previous chapters that a feasible method has been developed for the design of printed wiring boards. There are a number of improvements and alterations that should be made to the method to make it more useful to the industrial user, for whom it is intended. These include changes both to the basic algorithms and to the ways in which they are organised in the computer system. Many of the alterations are dependent on the type of hardware available to the user and the type of board layouts which he wishes to design.

11.1 Improvements to Topological Algorithm

The layouts illustrated in Chapter 10 show that the present topological algorithm produces graphs which are quite adequate for the type of board layouts considered. One possible improvement lies in the method of searching for paths to insert non-planar link branches into the pseudo-planar graph. At present a search is made from the regions around the start node of the branch to a region containing the target node. When several non-planar branches have a common start or target node, the search method can result in conductors following parallel paths under components as shown in Fig. 11.1(a). Examples of this can be seen in the crossing conductors of components R19 and R25 in Fig. 10.12.

The suggested improvement to the algorithm is that the search tree should also include regions which contain link branches that are already connected to the start or target node. The branch to be inserted may then be connected to one of these link branches so as to avoid parallel conductor paths under components to the

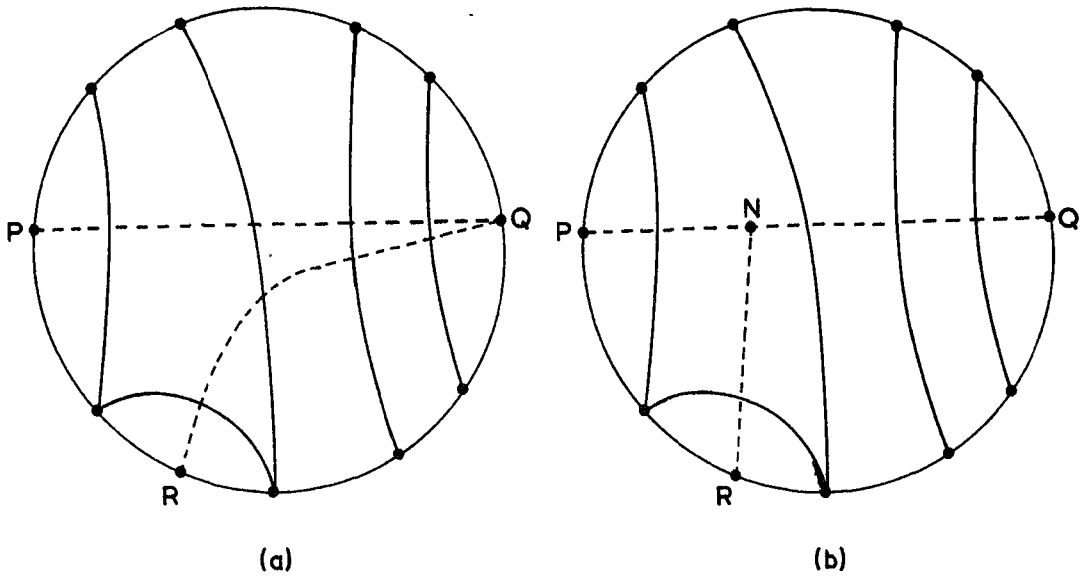


Fig. 11.1 Improvement to conductor routing

start or target node. The method is illustrated by Fig. 11.1(b). Link branch PQ is already in the graph and a search for path RQ yields a target region containing the branch PQ. In this target region PQ may be divided in two by the insertion of an extra node, N. The conductor path RN is then inserted as shown. This reduces the board area required for conductor routing and leaves more space under components for the insertion of further non-planar branches.

Non-planar component branches are inserted into the graph by splitting nodes and "hopping over" the conductors joining the two parts of each node. The number of nodes that can be split is limited by the physical dimensions of the component. At present, if the limit of nodes split is reached before a path to the target node has been found, the component is removed from the graph as non-planar. A possible improvement is to add a conductor branch to one end of the component at this stage. The path search may then be continued by crossing this conductor under other components.

Some components such as potentiometers, indicator lamps or test points may have to be mounted adjacent to one edge of the board for accessibility. At present the user can move such components towards the edge of the layout by interaction. He cannot guarantee to place them on the very edge of the layout due to the existing circuit topology. One way of solving this problem is to define a special pseudo branch in the graph which connects the component in question to one of the two end edge connector nodes. If the pseudo branch cannot be deleted as non-planar and cannot be crossed by conductors, the component will automatically be placed in the graph adjacent to the outside edge. The pseudo branch may later be removed when the pseudo planar graph is complete.

A limitation of the present program is that the connections of components to the edge connector pins have to be completely specified before the layout is started. It often occurs that a circuit may have several input or output nodes whose order of connection to the edge connector is not critical. In such cases the layout can frequently be improved and some non-planarities eliminated by re-arranging the order of nodes connected to the edge pins.

A possible solution to the edge connector problem is illustrated in Fig. 11.2. Three components, C1, C2 and C3 which form part of a circuit are to be connected to three edge connector pins, 1, 2 and 3. An arbitrary assignment of component-to-edge pin connections may produce non-planarities as shown in Fig. 11.2(a). This can be prevented by temporarily connecting the components and edge pins to a common node, N, as shown in Fig. 11.2(b). The

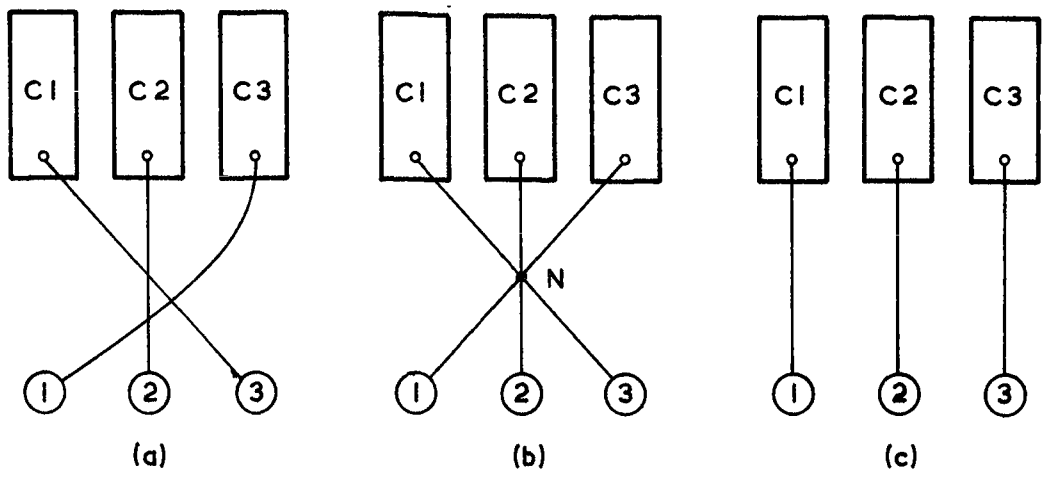


Fig. 11.2 Connection of components to edge connector

circuit data structure is then operated upon by the planarity algorithm.

The components and edge pins are connected to the same temporary node so their order of connection to it will be determined by the planarity algorithm. There will also be no non-planarity for that part of the circuit. When the planar graph is completed, the component-to-edge pin connections can be re-assigned and the temporary node removed as shown in Fig. 11.2(c). The same technique can also be used for integrated circuit components with multiple inputs.

The present planar graph algorithm is initialised with the assumption that the board to be laid out has an edge connector. It can be modified if necessary to deal with boards which have no edge connector. In this case, a search is made through the total graph to find a closed path of branches. This path is taken as the outside edge of the graph and all its branches are marked so that they cannot be crossed by conductors. The planarity algorithm then

proceeds as before. The laying out of such a graph is discussed in the next section.

11.2 Improvements to Layout Algorithm

Examination of the layouts in Chapter 10 shows that a number of improvements to the layout algorithm are possible. One of the more obvious improvements is to conductor paths such as those shown in Fig. 11.3(a). The reason for such paths has already been

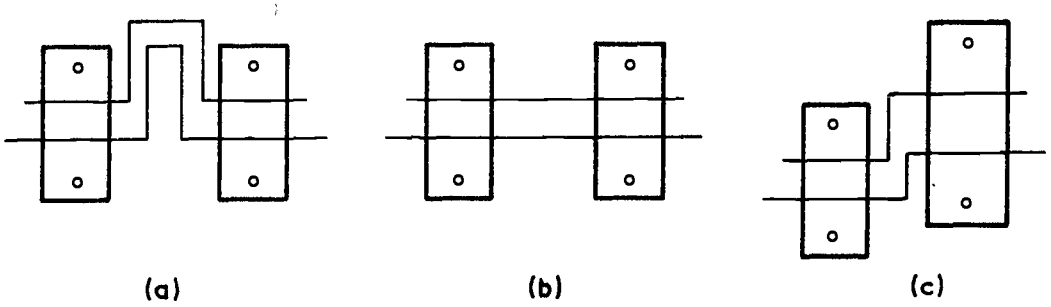


Fig. 11.3 Improvement to conductor paths

explained in Chapter 10.3.1 and the shortening of them is a straightforward task. The list of placed components is searched to ensure that none lie in the space between the two components. If the space is clear, the crossing conductors may then be routed as shown in Fig. 11.3(b). Ideally the two components should also be placed adjacent to each other so as to conserve board space. This involves further checking to ensure that the components are not at different levels or of different sizes such as those shown in Fig. 11.3(c).

A further improvement involves the routing of conductors under components. At present conductors are allowed under components only at crossing points. All other conductors are routed around the components as shown in Fig. 11.4(a). The proposed improvement is to

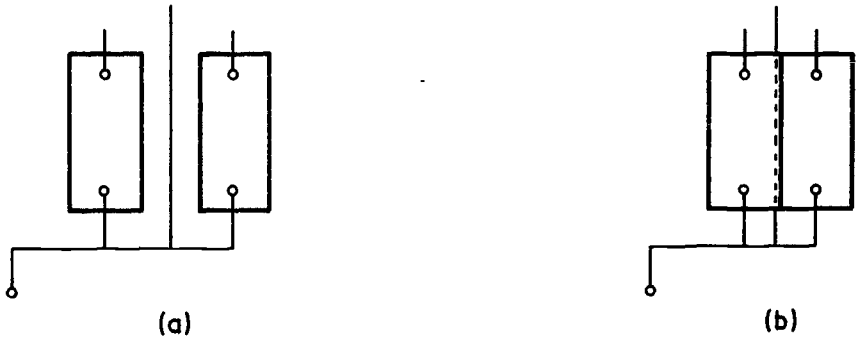


Fig. 11.4 Improvement to component spacing

allow conductor routing under components as shown in Fig. 11.4(b). This would enable the closer spacing of components and hence improve the board packing density. To implement this change it is necessary to compute the pin positions of all adjacent components so as to ensure that there is sufficient space for conductors between the pins. The user must decide in this case whether the saving in board space justifies the extra computation time required.

At present the layout program uses a standard conductor width for the whole layout. This is generally the way in which boards are designed but occasionally some conductors need to be of greater width to carry increased current. There are several possible solutions to this problem. One is to assign a conductor width to each circuit node at the data input stage. The corresponding width is then used during the layout construction. A second method is to define two or more parallel conductors between the appropriate points. During the layout stage the parallel paths are merged to form one conductor of the required width.

Many of the orthogonally routed conductor paths produced by the layout algorithm could be considerably reduced in length if diagonal routing were allowed. This has not been done at present

due to the problem of having to rigorously check the clearances between diagonally routed conductors whilst constructing a layout. One possible way of reducing conductor lengths is to allow interactions with conductor paths; this is discussed in the next section. Another possibility is to complete the layout then operate upon the data structure with a further program. This program would merely "round off" and shorten the existing conductor paths.

The problem of laying out boards with no edge connector has already been mentioned in the previous section. It may be dealt with in a straightforward manner. Instead of developing the initial working list of the layout from the edge connector nodes, the list is filled with components from the outside edge of the graph. These components are positioned in the first slot along the lower edge of the board. The layout algorithm may then proceed in the normal way to complete the layout.

A further extension to the layout algorithm would be to allow for obstacles in the layout. The obstacles could be such things as handles or fixing holes on the board. The program would require some form of "look ahead" capability when positioning components. It would ensure that conductors from the completed part of the layout could be routed around the obstacle and up to a higher level as shown in Fig. 11.5. If this feature were implemented it could also be used to deal with irregular shaped boards. The board shape would be defined as a rectangle with parts masked off by obstacles as shown in the example of Fig. 11.6.

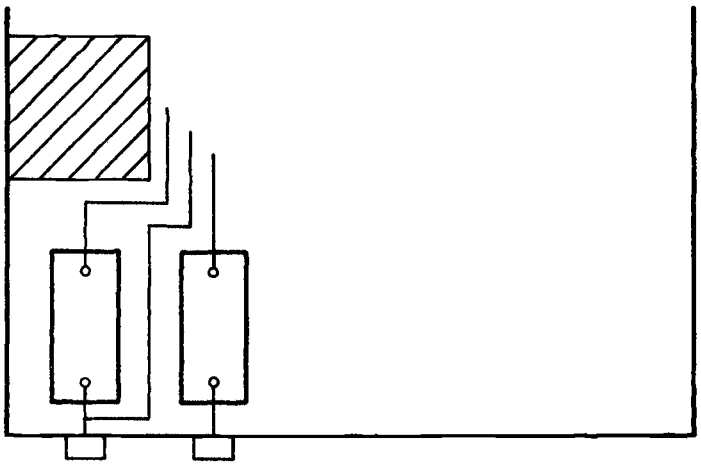


Fig. 11.5 Avoidance of obstacles on board

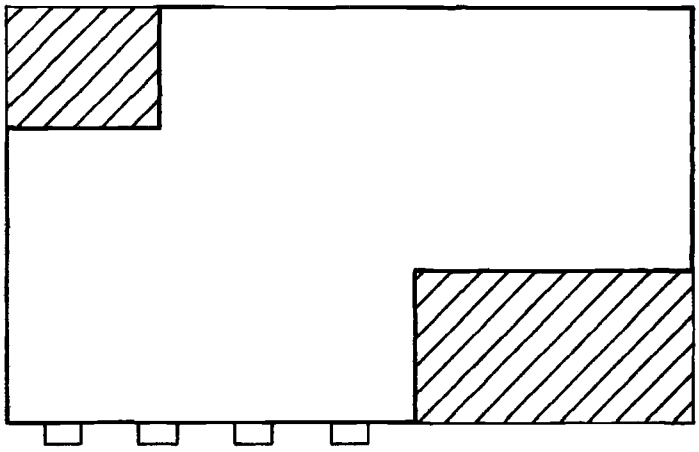


Fig. 11.6 Definition of irregular shaped board

Further improvements to the basic layout algorithm lie in the experience gained from using the interactive display. It is hoped that the insight gained from some of the interaction techniques can be incorporated into the automatic algorithm to improve its performance. One such technique already proposed is that components with many nodes and crossing conductors routed up to higher levels should themselves be moved up to higher slot levels. This would reduce the lengths of conductors attached to these components and improve the component packing density.

A further facility that could be added to the layout algorithm concerns the non-planar branches removed from the pseudo planar graph. At present a list of these branches, if any, is printed out when the pseudo planar graph is completed. The user later has to find the appropriate conductor paths in the layout to which wire jumpers may be connected. It would be a useful facility if the layout algorithm were to find the shortest distance between each pair of nodes to be connected and indicate the required paths for the wire jumpers.

11.3 Improvements to Interaction

There are a number of improvements that can be made to the interactive facilities available, some fairly simple and others of a more fundamental nature. One improvement immediately obvious to the user is the reduction in time needed to make a modification to the layout. Modifications near the top of circuit C in Fig. 10.12, for example, take from 10 to 15 seconds to be implemented. This is because the whole current layout has to be deleted then reconstructed up to the modification. This method has been used for the ease of programming although it is obviously not the most efficient way of using computer time. Methods for improving the interaction time are discussed in the next section.

It has been found from experience that a user may spend, say, half an hour interacting with a layout to obtain a satisfactory solution. On examining a hard copy of the display, a number of further improvements to the layout often become apparent. Before these improvements can be made in a later interaction session the whole procedure of modifying the layout has to be repeated. One

solution to this difficulty is to store the list of changes made to the layout on magnetic or paper tape. At the start of the next session, the list of changes can be read in and the whole modified layout built up automatically.

For large layouts there will be problems in displaying the whole board with sufficient detail to allow interaction. The most obvious solution is to "scissor" the display so that only a portion of the layout is seen, magnified to fill the whole screen. The display may then be considered as a "window" which can be moved by interaction over a much larger diagram of the whole layout. This facility should be implemented in the display software as it does not affect the basic layout algorithm.

The question of display software leads on to the problem of allowing interaction with the conductor paths of a layout. In a large layout there are many hundreds of conductor path change points. If every one is to be identifiable by the light pen it must be represented by a separate segment of display file. The storage requirement for the display file will then be considerably increased. A way out of this difficulty, again, is to scissor the display so that only a small portion of the layout is seen at any one time. This will result in a corresponding decrease in the length of the display file.

At present the user can alter the orientation of any component and put it into a particular slot by means of interaction. From then on the layout algorithm automatically positions the component in the slot, with clearances for adjacent conductors. All elements are placed in order from the left so that any spare

space is always on the right hand side of the slot.

A useful additional mode of interaction would be the ability to provide manual placement of components and conductors within a slot. This would allow the user to override the automatic algorithm. He could then place components towards the right hand side of the slot if desired, or pack them more closely by allowing overlap with conductors as shown in Fig. 11.4(b).

Experience of using interaction has shown that however much a layout is modified there are usually some long conductor paths that cannot be shortened. Component R12 in Figs. 10.6 and 10.7 is a good example; its connections must always be routed around the outside edge of the layout. The reason for this is that the circuit topology remains unchanged by interactive modifications to the layout. Again, it has already been noted that a short conductor path in the topological model does not necessarily give a short path in the layout.

The two points just noted could be improved by having a deeper level of interaction which would allow the user to modify the topology of the layout. Conductor paths could then be redefined so that although they crossed under more components their physical lengths were shorter. The modifications to the topology would have to be made by indicating which components in the layout were to be crossed by each new conductor path. Although it is possible to interact directly with the graph, it is difficult to visualise the layout from a diagram of the topological model. It would also be necessary to generate a display of the graph which is a considerable task in itself.

11.4 Improvements to Computer Organisation

There are a number of ways in which the layout program may be improved to make better use of the computer time and storage. These have not been implemented at present because of insufficient programming effort available and the fact that some of the computer facilities have been improved since the program was written. Several of the improvements discussed are intended for the present 4130 - PDP-7 system and might not be applicable to a different computer system.

An important improvement that could be made is in the time taken to make a modification to the layout by interaction. The 4130 computer now has a disc-based FORTRAN system which was not previously available. The disc system enables data to be stored and later retrieved at a high rate whilst the program is in operation.

The proposed modification is that the current state of the layout is recorded at the completion of each slot processing. When the contents of a slot are modified, the layout can be cut back to the previous slot level instead of having to rebuild the whole layout as at present. The problem still exists for PULL mode where a component can be pulled down below the previous slot level. In this case the reconstruction time could still be reduced by saving the state of the layout at selected lower levels.

When adding the contents of the latest slot to the layout, the current program completely regenerates the display file in order to show the latest slot. This means that as the working level of the layout increases and the display file becomes longer, the algorithm will take correspondingly more time to progress from one

slot to the next. The reason for using this method is that conductors are routed upwards through successive slots as continuous paths. With the present program there is no way of telling which part of a path has already been displayed. The whole display file is therefore deleted and then regenerated.

The display software allows the user to build up a display file as a series of segments. Later additions can be generated as separate segments and added onto the end of the existing display file. The use of markers or extra elements in the conductor path data structure could be used to indicate which parts of each conductor have already been displayed. Only additions to the display need then be generated and transmitted over the link, thus speeding up the layout program.

With the layout of very large circuits the program will run into problems of storage space for the data structure and display file. There are several possible solutions depending on the amount of storage space required. A number of elements in the data structure such as branch type markers and orientation markers are small integers so that several of them could be packed into one 24 bit computer word. This would decrease the storage requirements of the data structure and has already been done in the case of conductor path change points.

A more drastic approach would be to divide either the program or the data structure into several sections. The sections would be swapped between the disc and the core store during the running of the program. Only the required sections of program and data would then be held in the core store at any one time. Whichever

method is used, the program will take longer to run due to the extra unpacking of integers or the swapping of sections. The storage requirements of the display file can be alleviated by displaying only part of the layout at a time. This should be fairly easy to do if scissoring is available as part of the display software.

11.5 Extension to Double Sided Boards

The present version of the program deals with single sided boards as these have been most widely used up to now. Industry is making increasing use of double sided boards so it would be advantageous to extend the program to deal with such boards. Major changes would be necessary both to the topological and to the layout algorithms.

To extend the program to double sided boards, the topological data structure has to be modified so that conductors can be assigned to one or other side of the board. Component and pseudo branches have to be duplicated for the two sides of the board because every component pin hole appears on both sides of the board, at a defined distance from the remaining pins of its component. The suggested approach is that the graph of the circuit is operated on by the planarity algorithm to produce a planar graph for one side of the board. The resultant non-planar branches together with the components already in the first planar graph are then operated upon to produce a second planar graph for the other side of the board.

When the first planar graph is subtracted from the total graph some parts of the remaining graph may become disconnected from one another. The second pass of the planarity algorithm must therefore be able to deal with a graph which is composed of several

isolated subgraphs. Following the two passes of the planarity algorithm there may still be a few non-planar branches. These can be inserted into either of the two planar graphs by the methods described in Chapter 5.

The layout part of the algorithm also requires modification for double sided boards. Two base and two working lists are required for the two sides of the board. Only one slot profile list is required as components are placed on one side of the board only. The sizes and positions of successive slots are calculated from the slot profile list as before but in each slot, two base lists are developed simultaneously. Components and conductor paths are then placed in the slot in a similar manner to the present program. The display generation subroutine also requires modification so that the conductor paths on either side of the board can be distinguished. The modifications suggested will require considerable re-organisation of the program but should be rewarded by its increased usefulness.

11.6 Integration With an Industrial Environment

The current program has been developed to a state where it is possible to design a layout for a given circuit and set of components. The program output is in the form of one diagram which contains the essential information needed to construct a printed wiring board. There are a number of ways in which the program can be modified, mainly at the input and output stages, to provide more detailed and accurate data for the actual manufacture of boards. These modifications have not been implemented because they are dependent upon the individual user's requirements and computer system.

The first such modification that could be made to the program is the form of data input. At present there are possible sources of error in labelling the nodes on a circuit diagram, in extracting the component connection data from the diagram and in punching the data onto paper tape. A possible solution is to use the light pen and graphical display to draw the circuit diagram directly as input (32). At the same time the program can build up the corresponding topological data structure. This method has the advantage that any errors in the circuit description are far more easily detected from a graphical display than from a written table of data.

In some cases the user may initially employ a circuit analysis program to predict the performance of a circuit. The interconnection data of the circuit may then be fed directly into the layout program. This removes the possibility of errors at the input stage of the layout program. The user can be confident that the layout produced corresponds to the original circuit analysed. The present program also requires a component library to be read in for each board layout. Where the user has a data bank of standard components, this may replace the function of the component library.

On the output side of the layout program there are a number of possible improvements, depending on the equipment available to the user. The layout diagrams shown in Chapter 10 may be divided into two separate diagrams. One diagram would show the conductor paths so that a mask could be produced for etching the conductor pattern onto the printed wiring board. The second diagram would show the placement of components on the board and would be used when assembling the board.

The data structure representing the layout is flexible in use and may be processed by other programs to produce the type of output data required. One possibility is to produce a data tape for driving a mechanical plotter fitted with a light source and light sensitive paper. This would enable etching masks to be produced directly. A further possibility is to produce a data tape to operate a numerically-controlled machine for drilling the component pin holes in the board.

11.7 Discussion of General Points

As can be seen from the results in Chapter 10, the program produces a layout in which components are packed onto the board so as to make the best use of the available board space. The method is ideally suited to circuits which contain a number of different types of component such as resistors, capacitors, transistors and integrated circuits. The program in its present form is not suitable for circuits which contain mostly integrated circuits in a fixed matrix of positions on the board. Such boards place considerable constraints upon the algorithm and so are more appropriately laid out by one of the methods described in Chapter 2.

The data structures described in Chapters 6 and 9 use a variety of configurations, such as one-way lists, two-way lists and rings. These different types of structure are easily implemented by the use of the macro generator (Appendix A). It has been found that by matching a type of structure to the particular problem being solved, the programming is simplified and the storage space is efficiently used. This contrasts with other data structure systems in which only one configuration and hierarchy of elements is allowed.

The main disadvantage of using a mixed type of data structure is that programming errors, such as obtaining a pointer to a non-existent block, can be difficult to trace unless comprehensive checking procedures are used.

The current program has been developed for the design of printed wiring boards. Some of the algorithms used may be applied to other design problems. An obvious application is the design of integrated circuits where similar problems are encountered, although on a different physical scale. The relevant problems are those of arranging a number of components of varying sizes and shapes upon a plane surface and routing interconnections between them. Although it is not strictly necessary, it is preferable that the interconnection pattern is planar. The algorithms may also be used in other applications where it is necessary to design a set of paths between interconnected objects. One such possibility (14, 15) is the optimum layout of the rooms and corridors of a building.

Chapter 12 Conclusion

The results shown in earlier chapters indicate that a feasible method has been developed for the layout of printed wiring boards by computer. The layouts considered are of single sided boards containing discrete components of various sizes. The initial topological approach to the layout problem compares favourably with the more conventional method of component placement followed by conductor routing. As placement and routing are performed simultaneously, congestion of parts of the board by conductor paths is avoided. The automatic part of the layout algorithm produces useable layouts although it tends to form some long parallel conductor paths.

The results also show that significant improvements to a layout can be obtained by the use of interaction. The graphical display and light pen ensure close communication between the user and the layout program. Man-machine interaction thus enables the skill of the user to be combined with the speed and accuracy of the layout algorithm so as to rapidly produce a suitable layout. As the user only interacts with the highest level of the program, he is relieved of the detail of inserting conductor paths and checking component and conductor clearances. In addition, the algorithm ensures that the resultant layout corresponds exactly to the input data.

The algorithm produces results comparable with a manual layout method and in very much less time. It is thus suitable for use within an industrial environment. At present the results produced indicate the positions of components and the paths of conductors. Further improvements and modifications are required before production quality drawings may be output directly by computer.

Acknowledgements

The research work for this thesis was carried out in the Computer Aided Design Project of the Department of Computer Science. I am indebted to Dr. J.V. Oldfield who supervised the work and provided invaluable advice, assistance and criticism. Thanks are due to the other members of staff of the Project for facilities and assistance provided and to Mrs. P.S. Piper for typing this thesis. I am grateful to Jaroslav Jandos for very helpful ideas and profitable discussions on data structures and free storage systems.

The research work was supported by the Science Research Council and Associated Electrical Industries (New Parks) Ltd. on an Industrial Studentship. The interactive computing facilities were provided as part of a Science Research Council grant (Ref. B/SR/1718). Thanks are due also to the Department of Machine Intelligence and Perception for the use of the ICL 4130 computer.

Appendix A Use of Macro Processor

The ML/1 macro processor has been used for the programming of data structure operations within the FORTRAN language. It enables programs to be more easily written and understood and allows data structure definitions to be readily altered during program development. Only the facilities of ML/1 which have actually been used are described here. For a more detailed description of these and other facilities, the ML/1 Users Manual (4) should be consulted.

The ML/1 macro language provides general purpose macro processing facilities which can be used to process any piece of text. The processor requires an environment which defines the macro calls that are to be used. The input and processing of a piece of text is termed the evaluation of the text and the resultant output is termed the output text. The processor allows macro calls to appear anywhere in the text and allows any number of parameters to be associated with each call. The macro calls are of two types. Operation macros are defined as part of the system and are used to set up the environment. The three operation macros which have been used are MCSKIP, MCINS and MCDEF. Substitution macros are those defined by the user for specifying the way in which the text is to be evaluated.

MCSKIP

The operation macro MCSKIP allows parts of the source text to be skipped over during the evaluation of text. The macro defines a pair of delimiters, or skip names, which may appear in any number of places in the source text. The piece of text between each pair of delimiters may be copied over to the output text. Any macro

calls within this piece of text, however, will also be copied over and will not be evaluated.

The parameters of the MCSKIP macro are three optional characters followed by the skip names. The optional characters are:-

M - indicates that the two skip names are to be matched in pairs.

T - indicates that the text within the skip is to be copied to the output.

D - indicates that the skip names are also to be copied.

The MCSKIP macro has been used in this application as part of the definition of substitution macros, described later. The macro call used is:

MCSKIP MT, < >;

The skip names are < and > and the final semicolon is the delimiter of the macro call itself. This call defines a matched pair of skip names such that the text between each pair will be copied over to the output text but the skip names themselves will not be copied.

MCINS

When defining a substitution macro, it is necessary to indicate whereabouts in the replacement text the parameters of the macro call are to go. This is done by using an insert to indicate the relevant place for each parameter. The insert call itself has one associated parameter to define which parameter of the substitution macro call is to be inserted.

An insert is defined by use of the MCINS operation macro. The definition consists of an insert name followed by a delimiter which indicates the end of the insert parameter. As the insert

is associated with the definition of substitution macros, its insert name must be a string of characters which will not appear anywhere else in the source text.

As an example, the insert used for this particular application is described here. The insert is defined by:

```
MCINS  XX  .;
```

where XX is the insert name and . is its delimiter. An example of an insert call is then:

```
XX  A2.
```

where the parameter of the call is A2. The 2 indicates that the second parameter of a substitution macro call is to be inserted into the output text. The A indicates that all leading and trailing spaces around the parameter are to be suppressed.

MCDEF

The operation macro MCDEF enables the user to set up a substitution macro. The definition of a macro consists of three parts, a macro name, a delimiter structure and a replacement text. The macro name is the string of characters by which the macro call is identified. The delimiter structure defines the order and type of delimiters which separate the parameters of the macro call. The replacement text defines the output text and parameters which are to replace the original macro call.

An example of a typical macro definition is shown below. This particular macro is used to refer to the contents of the head of a data block. It is called with one parameter which is a pointer to the first word, or head, of the block. The data array to which the block belongs is called IRAY. The macro definition is thus:

```
MCDEF  HEAD(  )  AS  <IRAY(XX  A2.)>;
```

When the macro is called with one parameter, for example POINTER, it produces the following substitution:

```
HEAD(POINTER)  →  IRAY(POINTER)
```

Referring again to the definition above, the macro name of this definition is HEAD. The first and second delimiters of the definition are (and) respectively. The word AS then acts as a separator between the delimiter structure and the replacement text. By convention, each parameter of a macro call precedes its relevant delimiter so in the above example it is the second parameter which is to be inserted into the replacement text.

When the processor is actually evaluating a macro call, it first evaluates the arguments of the macro definition. This allows for the case in which the macro definition contains a call to another macro. For this reason, the replacement text of a macro is enclosed by a matched skip so that it is not evaluated during the definition of the macro.

Use of a Macro Processor

To use the macro processor, a paper tape is first prepared containing all the macro definitions which will be required. The processor program tape is then read into the PDP-7 computer, followed by the macro definition tape. This sets up the environment so that the processor is ready to evaluate any number of source tapes. As each source tape is read in and evaluated, the source text is copied over to an output paper tape until a macro name is identified. The appropriate replacement text is output then the evaluation of source text continues.

Three examples of the use of macros are shown below. The first example is a macro that refers to the first node to which a branch is connected. The macro definition is:

```
MCDEF  BNODE1(  )  AS  <IRAY(XX  A2.+1)>;
```

The resultant substitution is:

```
BNODE1 (BPTR) →  IRAY(BPTR+1)
```

The second example is a macro which refers to the X co-ordinate of the Nth pin of a master component block in the component library. Its definition is:

```
MCDEF  MCORDX( , )  AS  <IRAY(XX A2.+11+XX A3.+XX A3.)>;
```

and its resultant substitution is:

```
MCORDX(MPTR,N)  →  IRAY(MPTR+11+N+N)
```

The third example shows how the marker in the head of a component branch block may be defined as an integer number. The definition is:

```
MCDEF  MARKCB  AS  <1000000>;
```

A statement containing two macro calls:

```
HEAD (BRANCH) = MARKCB
```

will result in the following FORTRAN statement:

```
IRAY(BRANCH) = 1000000
```

All the operations on the data structure for the layout algorithm are defined in a similar manner by the use of macro calls.

Appendix B Display Software

The interactive display software enables the user to generate a graphical display and to use it for interaction with his program. The software is organised into two parts, one part residing in the ICL 4130 computer and the other part in the PDP-7 computer. The set of subroutines in the 4130 may be called from the user's FORTRAN program. These subroutines generate and operate upon a display file which is held in a large array. When a new display file is generated, or modifications are made to an existing display file, the relevant parts of the file are transmitted over the high speed link to the PDP-7 computer.

The software in the PDP-7 includes a Link Executive program which controls the data transfers in both directions over the link. Whenever a display file or modification is received from the 4130, the display file in the PDP-7 is immediately updated so that the change is seen on the Type 340 display. The software also services the display tracking cross and handles interrupts from the light pen and Teletype keyboard. When an interrupt occurs, the relevant data is assembled into a four-word attention block. This block may then be transmitted back over the link to the 4130 when requested by a call from the user's FORTRAN program.

The FORTRAN display and interaction subroutines associated with the user's program are described below. Only those facilities which have actually been used for the layout algorithm are described. For a more detailed description of these and other facilities, the system description (11) should be consulted.

1. Generation of Display File

The first subroutine that must be called before generating a display file has the calling sequence:

```
CALL DEFPIC(IFILE, LIMIT)
```

The subroutine passes over to the display software the name of the array which is to hold the display file, IFILE. The maximum allowable size of this array is defined by the value of LIMIT.
The second subroutine which must be called has the calling sequence:

```
CALL SENTER
```

This subroutine initialises all the display subroutines and causes a set of character definitions to be read in from magnetic tape. The characters are defined as display subroutines because no character generator is available.

The basic subroutine for plotting points on the display has the calling sequence:

```
CALL MOVETO(IX,IY,VIS,ISCALE,INTENS)
```

where IX and IY are the required co-ordinates of the point. VIS is a logical variable which determines whether or not the point is visible. The value of ISCALE sets the scale of the following display file and the value of INTENS sets the display intensity.

The basic subroutine for drawing straight lines has the calling sequence:

```
CALL LINE(IDELX,IDELY,VIS)
```

where IDELX and IDELY are the required X and Y displacements from the current beam position. The logical variable VIS determines whether or not the line is visible.

The software has facilities for generating and calling display subroutines. A display subroutine may be generated at any point in the display file but its definition must be complete before a call is made to the subroutine. Every display subroutine is assigned a unique system name by the software. The system name is, in fact, the index of the display file array at the first element of the subroutine.

A display subroutine definition is commenced by the calling sequence:

```
CALL DEFSUB(NAMSUB)
```

The value of the variable NAMSUB is set to the system name of the subroutine by the display software. The lines and characters defining the subroutine are then generated by calls to the appropriate routines. The definition of the display subroutine is concluded by the calling sequence:

```
CALL ENDSUB(NAMSUB)
```

Whenever an instance of the display subroutine is then required, it is obtained by calling:

```
CALL CALSUB(NAMSUB)
```

Two functions are available for generating alphanumeric characters. The first one displays a single character and is called by:

```
NAME = CHAR41(NCODE)
```

where NCODE is an integer code for the character to be plotted. The character is defined as a display subroutine so its system name is assigned to the variable NAME. This means that copies of the character can then be displayed by calling it as a display subroutine with the parameter NAME. The second function enables a string of

characters to be plotted. It is called by:

NAME = TEXT(IARR,N)

where NAME serves the same purpose as before. IARR is the name of an array which contains the codes for the characters to be plotted, packed four to a word. N is the number of characters to be plotted.

When the whole display file has been generated it is terminated by the calling sequence:

CALL DEFPIC(IFILE,MEDIUM)

where IFILE is the display file array. MEDIUM is an integer variable, the value of which determines whether the display file is transmitted over the link to the PDP-7 or is punched out on paper tape.

2. Display File Editing

The display software enables the display file to be divided into a number of segments. These segments are linked together in a simple list so that each segment may be displayed in turn. When extra segments are added to the list, or existing segments deleted only the differences in the display file are transmitted to the PDP-7. This considerably reduces the amount of data sent over the link when making small changes to a large display file.

Every segment has a three word header block followed by a section of display file. The header block contains a pointer to the next segment in the list, a system name and a user name. The system name is merely the array index of the first word of the segment. The user name is an integer value which the user may associate with the segment.

The display software automatically creates the first segment at the start of the display file. When another segment is required,

it is obtained by calling:

```
ISEG = NEWSEG(LABEL)
```

ISEG is assigned the system name of the new segment and LABEL is its user name. Whenever a new segment is opened, the previous segment is automatically terminated.

An alternative method of creating a segment is by calling the function:

```
ISEG = INSTAT(IX,IY,NAMSUB,LABEL)
```

This function is used to create an instance of the display subroutine NAMSUB at the co-ordinates IX and IY. ISEG is assigned the system name of the segment and LABEL is its user name.

After any segment has been defined, its display scale or its intensity or both may be altered by calling:

```
CALL CHINTS(ISEG,ISCALE,INTENS)
```

ISEG is the system name of the segment and ISCALE and INTENS are the new values of scale and intensity respectively.

Segments can be deleted from the display in a number of ways. A segment may be temporarily deleted by the calling sequence:

```
CALL REMOVE(ISEG)
```

where ISEG is the system name of the segment. The segment disappears from the visible display although it remains in the display sequence. The segment may be restored by calling:

```
CALL RESTOR(ISEG)
```

The second method of deleting a segment removes the segment permanently from the display file so that the corresponding array space may be used again. Any display subroutines in the segment are thus deleted as well. All calls to these subroutines are therefore

removed from the remainder of the display file by the display software. The required calling sequence is:

```
CALL CANCEL(ISEG)
```

The third method of deleting a segment is by the calling sequence:

```
CALL DELETE(ISEG)
```

This removes the segment from the display sequence although it remains in the display file array. The method is used for segments which contain only subroutine or character definitions that are not to appear in the display until called from later segments.

3. Light Pen and Keyboard Interaction

When using interactive computer graphics, the light pen and Teletype keyboard on the PDP-7 computer are the means by which the user communicates with his program. The light pen enables parts of the display file to be identified and the keyboard enables single characters to be sent to a FORTRAN program in the 4130 computer. As the PDP-7 computer cannot directly interrupt the FORTRAN program in the 4130, it stores an attention block. This block may then be read from the FORTRAN program to determine which device in the PDP-7 caused an interrupt.

Before the light pen or Teletype can cause an interrupt they have to be enabled. This is done by the following call from the user's FORTRAN program:

```
CALL ENABLE(I)
```

The parameter I is an integer whose value determines whether the light pen or the Teletype keyboard is to be enabled. A similar subroutine call allows the user to disable either device at any stage of the interactive program.

Every segment of the display may be made either sensitive or non-sensitive to the light pen. This allows the user to organise the display file so that light pen interrupts are obtained only from the relevant parts of the display. A segment is made sensitive to the light pen by the call:

```
CALL MSSLP(ISEG)
```

Pointing the light pen at a display segment will thus cause an interrupt only if the segment is made sensitive and the light pen is enabled.

The user's FORTRAN program can be made to wait for an interactive operation by the call:

```
CALL ACTION(IRAY)
```

The program waits in a loop until an attention block is ready in the PDP-7 computer. The contents of the block are then read into the four-word array IRAY. The first word of this array indicates whether the attention was caused by the light pen or the Teletype keyboard. The second word gives the system name of the segment in which the light pen hit occurred, or the six bit code for the character entered on the keyboard. The following two words give the X and Y co-ordinates of the light pen hit if appropriate.

At some stages of the user's interactive program it may be desirable to remove any redundant light pen hits before proceeding to the next stage. This is effected by the call:

```
CALL ATKILL
```

The call causes the attention mechanism to be reset so that all attention blocks waiting in the PDP-7 are cancelled.

4. Tracking Cross Routines

The display software allows a tracking cross to be used and its co-ordinates to be read to the user's FORTRAN program. The tracking cross is made to appear on the display by calling:

```
CALL TRSET(IX,IY)
```

The parameters IX and IY are the co-ordinates of the position at which the cross is to appear. The tracking cross may then be tracked by the light pen without further attention from the 4130 computer.

The current co-ordinates of the tracking cross may be read at any time by the call:

```
CALL TRACK(IX,IY,ISTOP)
```

The parameter ISTOP is an integer variable which indicates whether the PDP-7 has an attention block waiting. When the tracking cross is no longer required, it can be removed from the display by calling:

```
CALL TRKILL
```

The subroutines described here provide the user with fairly sophisticated facilities for interactive programming. A display file can readily be constructed and modified by interaction. The segmentation and naming system then enables the user program to rapidly determine which part of the display was seen by the light pen.

Glossary of Terms

Base limits	Two variables associated with a base node. They store the extreme X co-ordinates of the node during conductor routing.
Base list	A list of the nodes and conductors along the bottom edge of a slot, used during the construction of a layout.
Block	A group of consecutive words of the data storage array, used to store information on an element of the graph or layout.
Board	A thin board of insulating material which supports the components of a layout. A pattern of copper conductors is etched onto one or both sides of the board to interconnect the components.
Bound branches	Two branches associated with each part of a node during the construction of the layout. They indicate those parts of the node which have already been placed in the layout.
Branch	An element of a graph which interconnects a pair of nodes, sometimes termed an "edge" of the graph.
Branch component	A component with two connecting wires or pins. It is represented in the graph by a component branch.
Branch segment	A component, pseudo or link branch is divided into two branch segments when crossed by another branch. The division of branches is caused by the insertion of non-planar branches into the pseudo-planar graph.

Bridge branch	A branch which provides the only connection between a subset of the planar graph and the remainder of the graph.
Circuit (electrical)	A specified set of components and their interconnections which performs an electrical function on the signals applied to it.
Circuit (graphical)	A set of branches which form a closed path in the graph.
Circuit node	A point of common electrical connection of two or more components.
Component	An element of the electrical circuit, such as a resistor or a transistor.
Component branch	See Branch component.
Component pin	A terminal wire or connection point of a component which passes through a hole in the printed wiring board and connects with a conductor.
Conductor	A copper track formed onto the printed wiring board which connects one part of the circuit to another.
Conductor branch	A branch representing a conductor which connects two circuit nodes. It is formed when splitting a node into two parts during the insertion of a non-planar branch.
Data structure	The system of data blocks and pointers which is used to represent the graph and layout within the computer store and which holds details of the component library.

Development	During the construction of the layout, a node in the base list is developed by creating a list of all the components and conductors which could be directly connected to it.
Discrete component	A component which performs a single electrical function, such as a resistor or a capacitor. A number of these components must be interconnected in order to construct a circuit, as opposed to an integrated circuit component in which a complete circuit is included within one package.
Double sided board	A board which has a conductor pattern on both sides.
Edge connector	A set of gold plated conductors along one edge of the board, perpendicular to that edge, to which the external connections of the layout are brought. The edge connector plugs into a multiway socket to make contact with external signals and power supplies.
Edge pin	One of the conductors of the edge connector.
Free region	Used during the construction of the planar graph. It contains all those nodes and branches which have not yet been defined as part of the graph.
Graphical display	The visual display of a layout, plotted on a cathode ray tube.
Interaction	The close communication between a computer program and the user, whilst the program is running.
Layout	The arrangement of component positions and conductor paths on a printed wiring board.

Link branch	A branch which connects a subgraph node to its corresponding circuit node. It is represented physically by a length of conductor connected to one pin of a subgraph component.
Master component	An item in the component library which describes all the common characteristics of a particular group of components.
Node splitting	The process by which a node is divided into two parts, connected by a conductor branch, so that a non-planar component branch can be inserted into the graph.
Orthogonal routing	The method of routing conductors in which all parts of every conductor lie parallel with either axis of a rectangular board.
Part of a node	A conductor path which is connected to one or more components of a given circuit node. The node may exist in several parts during the construction of a layout.
Planar graph	A graph which may be drawn on a plane in such a way that its branches intersect only at their end points. Planarity is an intrinsic property of a graph and so is independent of any geometrical representation of the graph.
Pointer	Used to indicate the interconnections between blocks in the data structure. A pointer to a block is a variable containing the array index of the first element of that block.

Pseudo branch	A branch connected between two pins of a subgraph component or between two edge connector nodes. It is used to represent the physical distance between two nodes so as to limit the number of conductors passing between the nodes.
Pseudo planar graph	A graph which represents all the components and interconnections of a layout. It represents a planar set of conductor paths even though there are a number of component/conductor branch crossings.
Region	An area within a planar graph bounded by a closed path of branches.
Routing	The process of constructing a conductor path between two points in the layout.
Single sided board	A board which has a conductor pattern on one side only. The components are mounted on the opposite side of the board.
Slot	An area of the board, bounded on three sides by placed components or board edges. It is used as a working area for constructing a further part of the layout.
Source pin	The component pin which connects a component to the base node from which it was developed.
Subgraph component	A component with more than two connecting wires or pins. It is represented in the graph by a set of subgraph nodes, pseudo branches and link branches.

Subgraph node	A node representing one pin of a subgraph component.
Through plated hole	A copper-lined hole through a board which makes a connection between conductors on the two sides of a double sided board.
Tie block	A two-element block which associates a branch segment with its two adjacent regions.
Topological model	The pseudo-planar graph which represents the order of connection of all the components and conductor paths in a layout.
Tree	An ordered hierarchy of nodes and branches, used to record the progress of a search through the graph.
Wire jumper	An insulated piece of wire connected into a single sided board layout. It enables two conductor paths to be crossed without interconnection.
Working list	A list used during the construction of a layout. It holds data which is processed to determine the contents of a slot and the physical co-ordinates of these contents.

References

(1) W. Bader, "The topological problem of the printed circuit board and its solution", Archiv fur Electrotechnik, vol. 49, no. 1, pp. 2-12, 1964.

(2) J.W. Brackett, A.C. Kilgour, J.V. Oldfield, "Fortran package for generating a PDP-7 display file", CAD project report no. CAD-R-14, University of Edinburgh, Nov. 1967.

(3) M.A. Breuer, "General survey of design automation of digital computers", Proc. IEEE, vol. 54, no. 12, pp. 1708-1721, Dec. 1966.

(4) P.J. Brown, "ML/1 Users Manual", University Mathematical Lab., Cambridge, June 1967.

(5) R.G. Busacker & T.L. Saaty, "Finite graphs and networks", McGraw Hill Book Co., 1965.

(6) P.W. Case et al, "Solid logic design automation", IBM Journal of Research & Development, vol. 8, no. 2, pp. 127-140, April 1964.

(7) G.V. Dunne, "The design of printed circuit layouts by computer", Proc. 3rd. Australian Computer Conference, Canberra, pp. 419-423, May 1966.

(8) G.J. Fisher & O. Wing, "Computer recognition and extraction of planar graphs from the incidence matrix", Trans. IEEE, CT-13, no. 2, pp. 154-163, June 1966.

(9) C.J. Fisk et al, "ACCEL: Automated circuit card etching layout", Proc. IEEE, vol. 55, no. 11, pp. 1971-1982, Nov. 1967.

(10) W.P. Heising, "History and summary of FORTRAN standardisation development for the ASA", Comm. ACM, vol. 7, no. 10, pp. 590-625, Oct. 1964.

(11) A.C. Kilgour, M.D. Brown, "SPINDLE ; a system permitting interactive display list editing" CAD project report no. CAD-R-55, University of Edinburgh, June 1969.

(12) A.C. Kilgour, "Program to plot a display file on the incremental plotter", CAD project report no. CAD-R-37, University of Edinburgh, Dec. 1968.

(13) V.R. Kodres, H.E. Lippmann, "STL board layout", IBM technical report no. TR 00.1010, March 1964.

(14) M. Krejcirik, "Computer aided plant layout", Computer Aided Design, pp. 7-19, Autumn 1969.

(15) M. Krejcirik, "Computer aided building layout", IFIP Congress, Edinburgh, pp. I126-I130, August 1968.

(16) G. Kuratowski, "Sur le probleme des courves gauches en topologie", Fundam Math. vol. 15, pp. 271-283, 1930.

(17) C.A. Lang, J.C. Gray, "ASP - A ring implemented associative structure package", Comm. ACM, vol. 11, no. 8, pp. 550-555, August 1968.

(18) S.E. Lass, "Automated printed circuit routing with a stepping aperture", Comm. ACM, vol. 12, no. 5, pp. 262-265, May 1969.

(19) C.Y. Lee, "An algorithm for path connections and its applications", IRE Trans. on Electronic Computers, vol. 10, no. 3, pp. 346-365, Sept. 1961.

(20) D.F.A. Leevers, "The use of a graphical display in the automatic design of printed circuit boards", International Conference on CAD, Southampton, IEE Conference publication no. 51, pp. 11-20, April 1969.

(21) S. MacLane, "A combinatorial condition for planar graphs", Fundam. Math. vol. 28, pp. 22-32, 1937.

(22) J.S. Mamelak, "The placement of computer logic modules", Journal ACM, vol. 13, no. 4, pp. 615-629, Oct. 1966.

(23) D.D. McCracken, "A guide to FORTRAN IV programming", John Wiley & Sons, Inc. 1965.

(24) K. Mikami, K. Tabuchi, "A computer program for optimal routing of printed circuit conductors", IFIP Congress, Edinburgh, pp. H47-H50, August 1968.

(25) W.M. Newman, "The ASP-7 ring-structure processor", Computer Technology Group Report 67/8, Imperial College, Oct. 1967.

(26) T.A.J. Nicholson, "A permutation procedure for minimising the number of crossings in a network", Proc. IEE, vol. 115, no. 1, pp. 21-26, Jan. 1968.

(27) D.T. Ross, "A generalised technique for symbol manipulation and numerical calculation", Comm. ACM, vol. 4, pp. 147-150, 1961.

(28) D.T. Ross, "The AED free storage package", Comm. ACM, vol. 10, no. 8, p. 481 August 1967.

(29) T. Rowley, "Further topology associated with automatic layout of printed wiring", AEI internal report no. NP.g.12, Sept. 1967.

(30) R.A. Rutman, "An algorithm for placement of interconnected elements based on minimum wire length", Proc. SJCC, AFIPS, vol. 25, pp. 477-491, 1964.

(31) H. Whitney, "Non-separable and planar graphs", Trans. Amer. Math. Soc., vol. 34, pp. 339-362, 1932.

(32) N.E. Wiseman et al, "PIXIE - a new approach to graphical man-machine communication", International Conference on CAD, Southampton, IEE Conference publication no. 51, pp. 463-471, April 1969.